GraCE: Unlocking CUDA Graphs with Compiler Support for ML Workloads

Abhishek Ghosh[1], Ajay Nayak[1], Ashish Panwar[2], Arkaprava Basu[1]

[1]*Indian Institute of Science* [2]*Microsoft Research India*

Mind the Gap Between Hardware & Software

Slowing Transistor Technology → No Free Lunch for Software



Mind the Gap Between Hardware & Software

Software-Managed Hardware Features



Mind the Gap Between Hardware & Software

High level ML Programming Frameworks

PyTorch

TensorFlow

Software-Managed Hardware Features



Mind the Gap Between Hardware & Software

High level ML Programming Frameworks

PyTorch

TensorFlow



Software-Managed Hardware Features



Widening semantic gap

Bridging the Gap Between Hardware & Software

High level ML Programming Frameworks

PyTorch

TensorFlow

Software-Managed Hardware Features

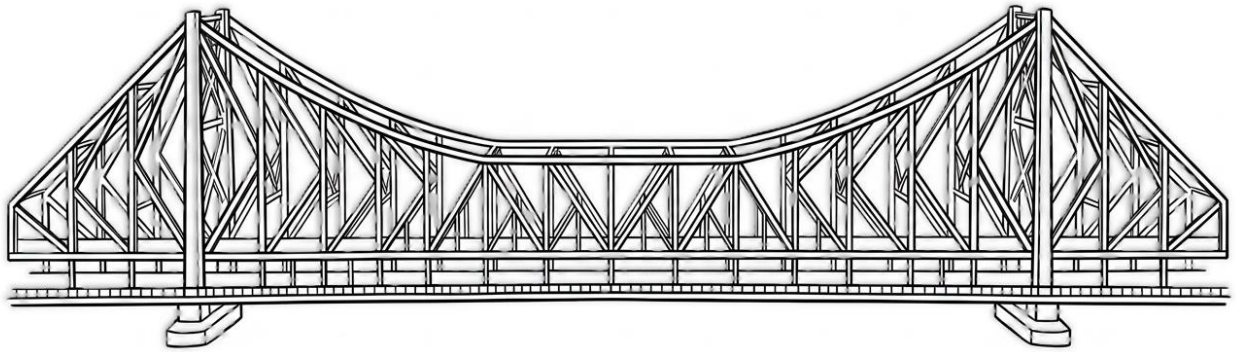


Bridging the Gap Between Hardware & Software

High level ML Programming Frameworks

PyTorch

TensorFlow



Software-Managed Hardware Features

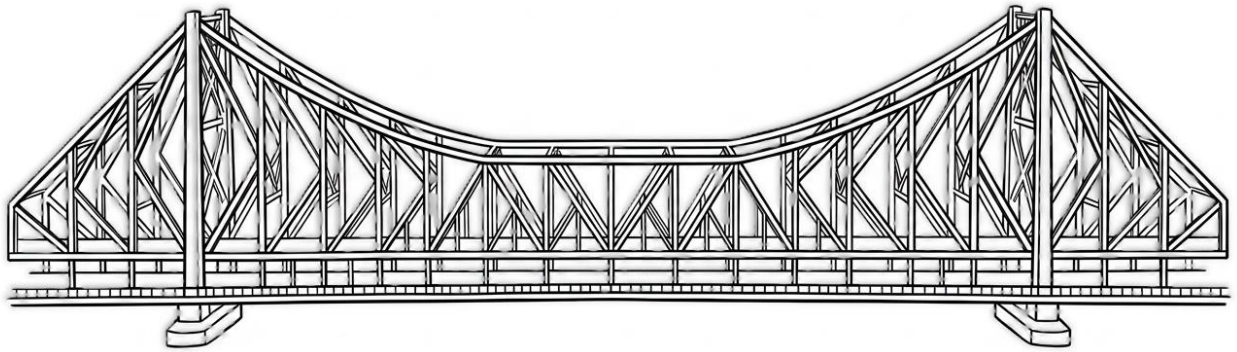


Bridging the Gap Between Hardware & Software

High level ML Programming Frameworks

PyTorch

TensorFlow



Software-Managed Hardware Features



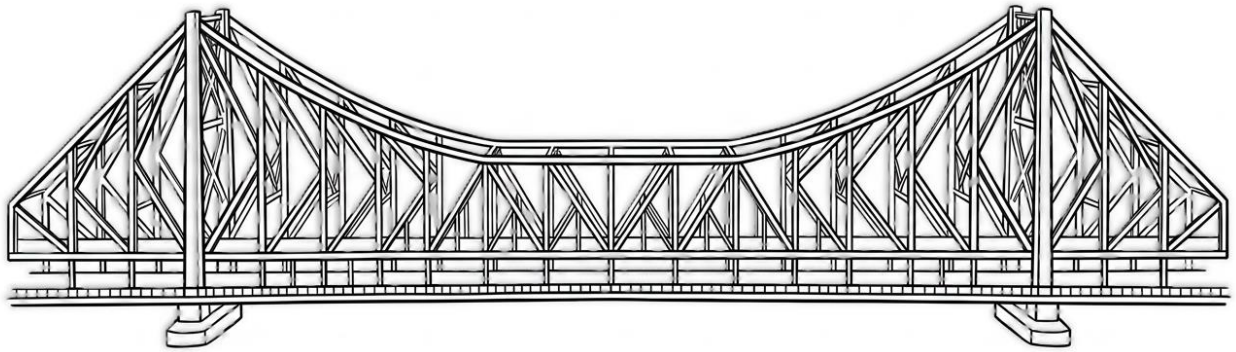
Must NOT need hardware changes

Bridging the Gap Between Hardware & Software

High level ML Programming Frameworks

PyTorch

TensorFlow



Software-Managed Hardware Features



Must NOT need application changes

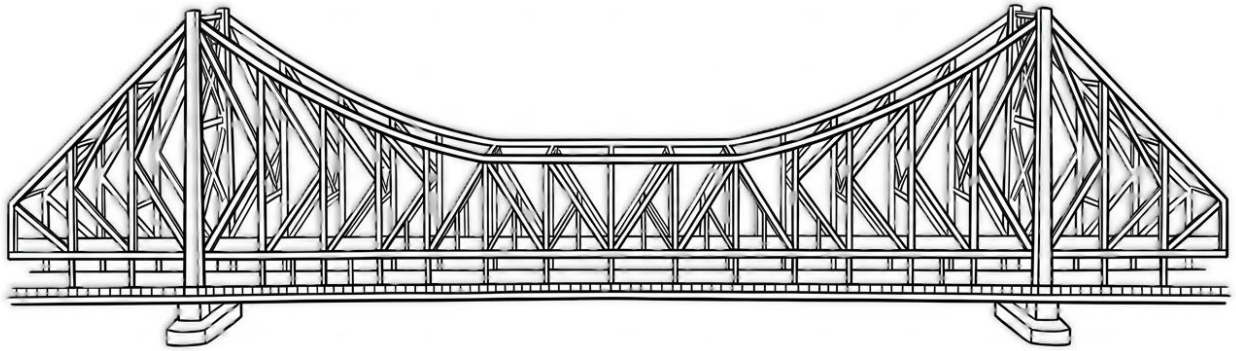
Must NOT need hardware changes

Bridging the Gap Between Hardware & Software

High level ML Programming Frameworks

PyTorch

TensorFlow



Focus: Hardware-aware Middleware

Software-Managed Hardware Features



Must NOT need application changes

Must NOT need hardware changes

CPU Launches Kernels on the GPU



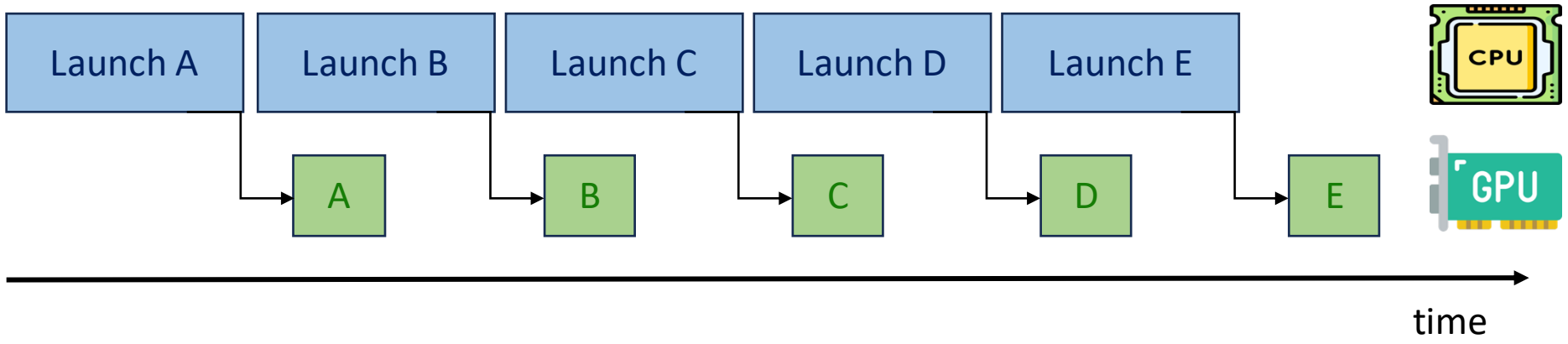
CPU Launches Kernels on the GPU



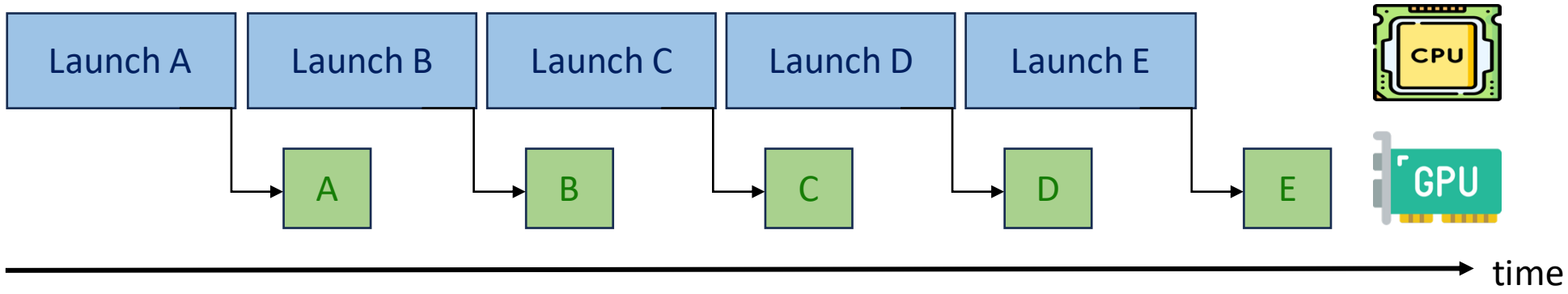
CPU Launches Kernels on the GPU



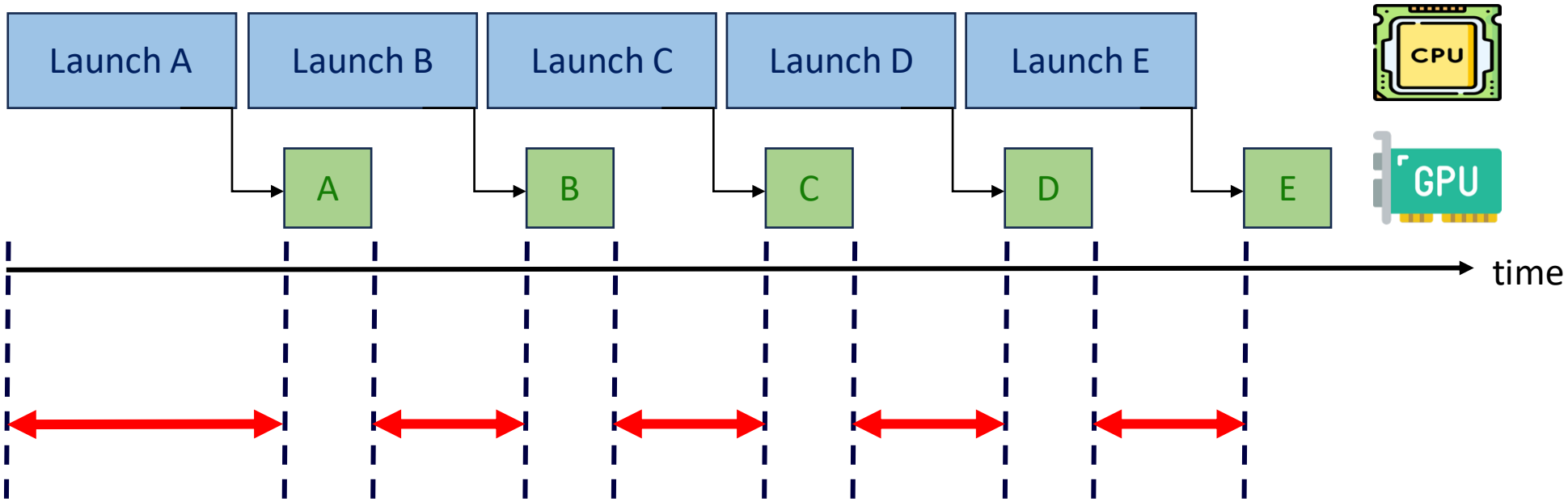
CPU Launches Kernels on the GPU



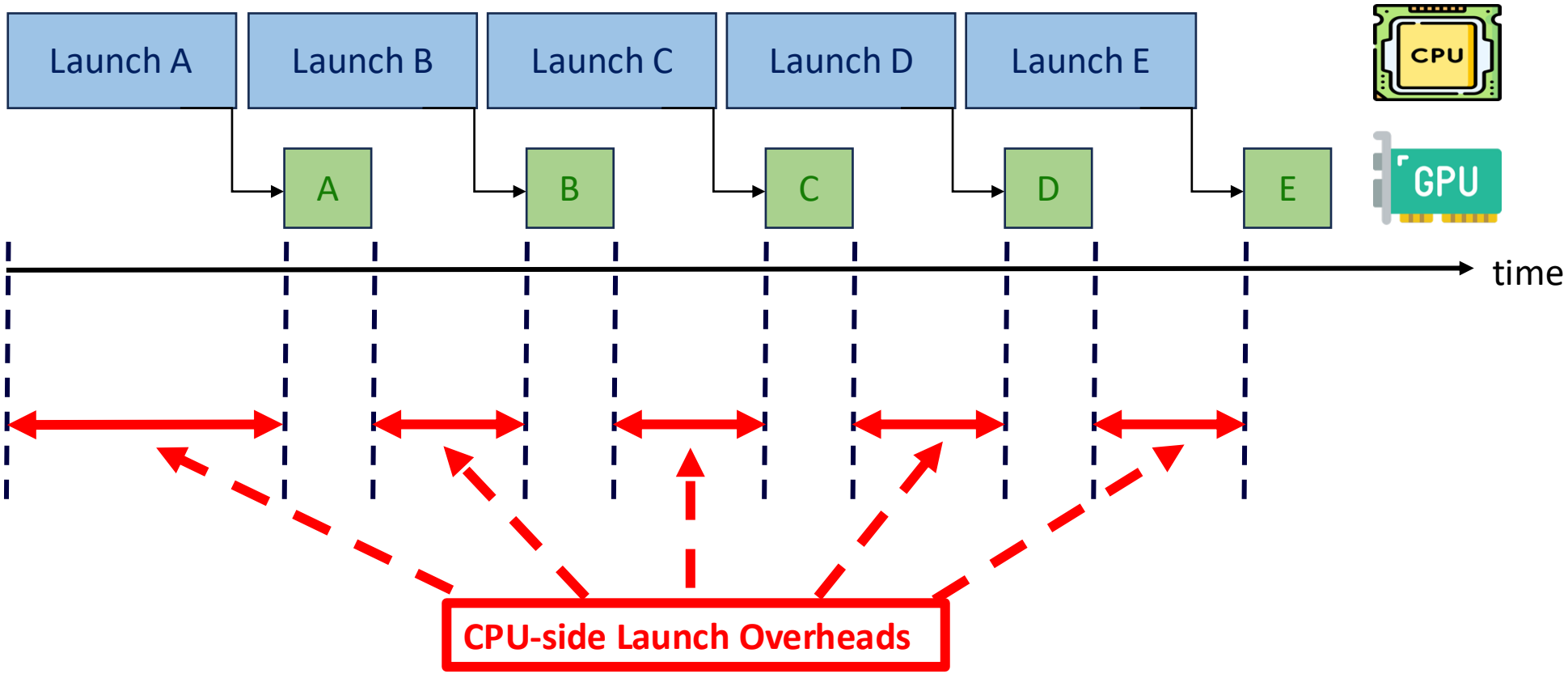
GPU Kernel Launch Overhead Can Dominate



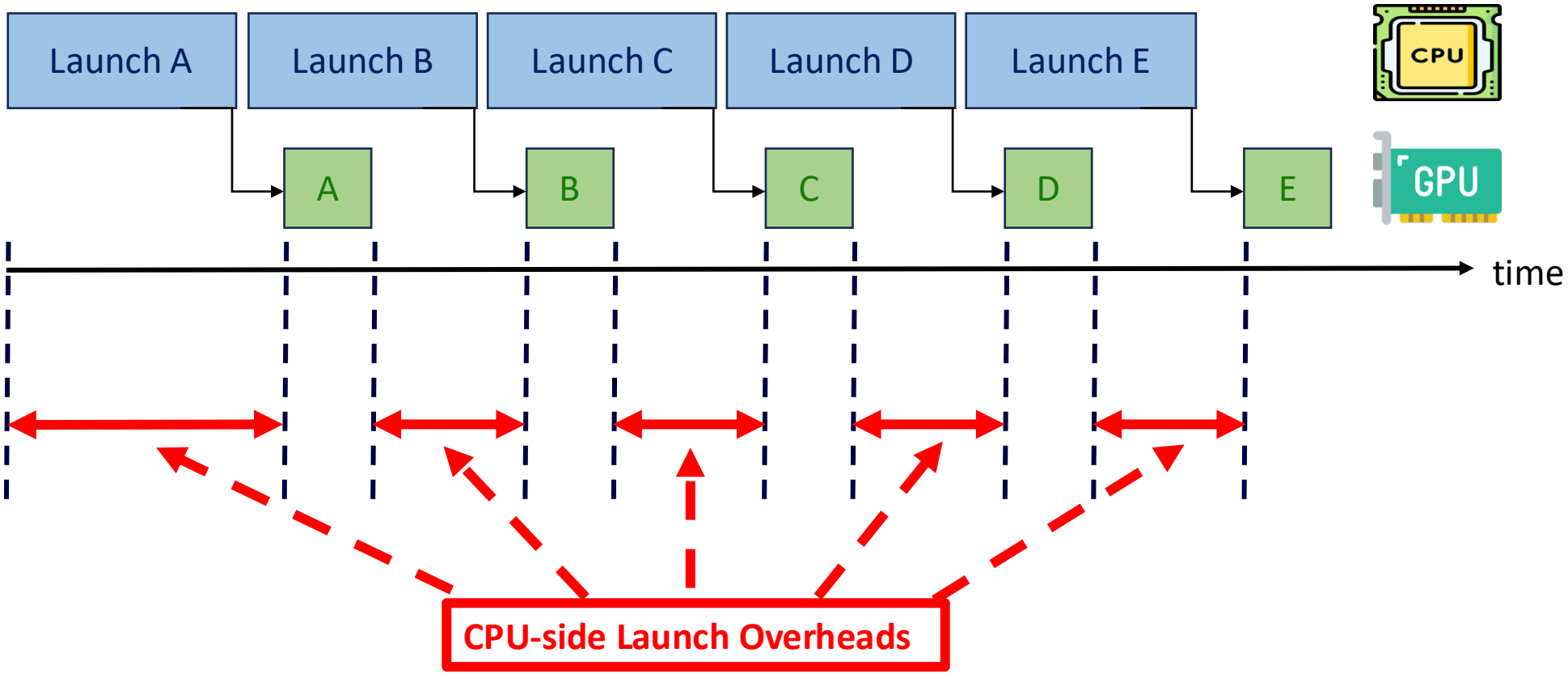
GPU Kernel Launch Overhead Can Dominate



GPU Kernel Launch Overhead Can Dominate

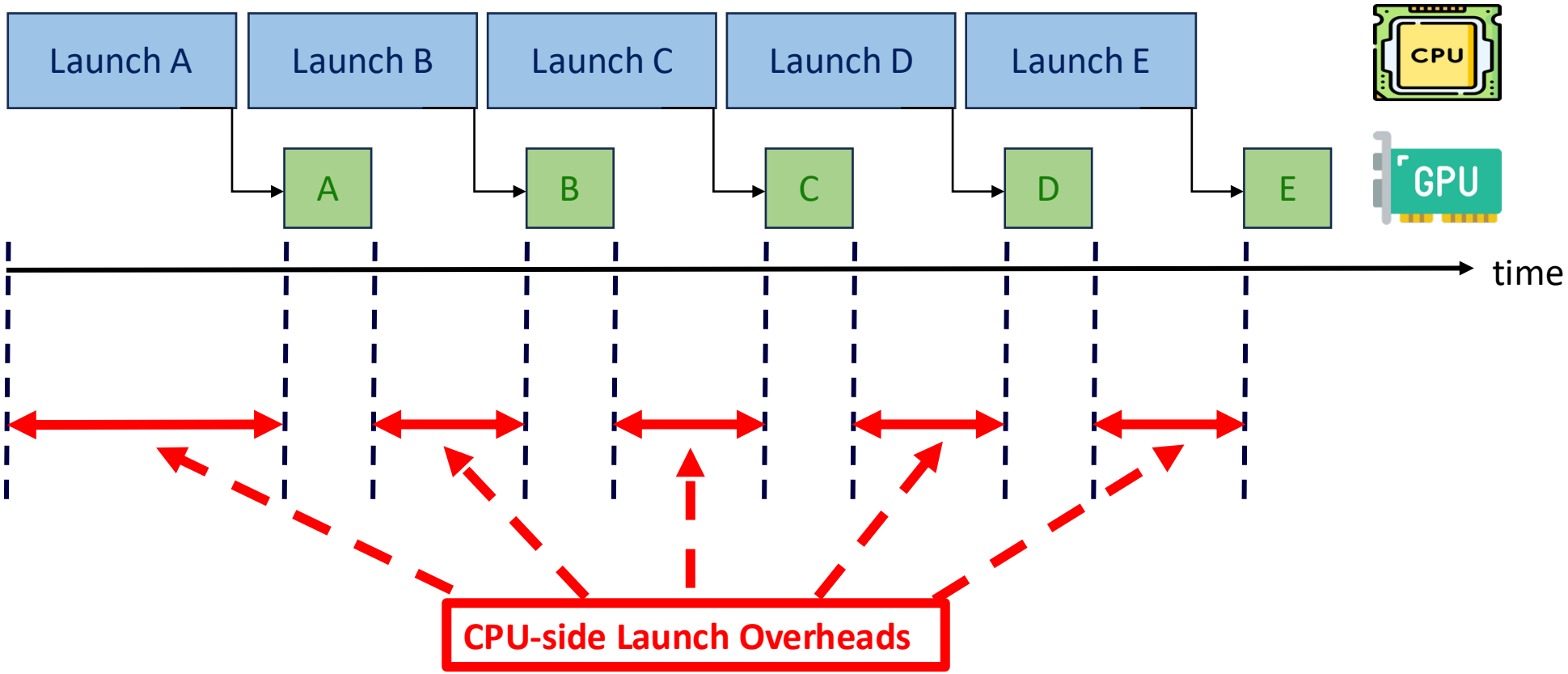


GPU Kernel Launch Overhead Can Dominate



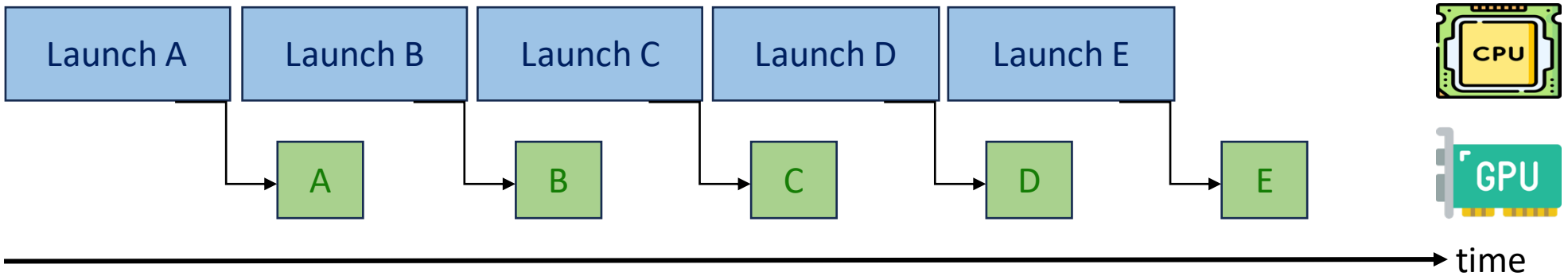
When kernel runtime is short, execution time is dominated by CPU launch cost

GPU Kernel Launch Overhead Can Dominate

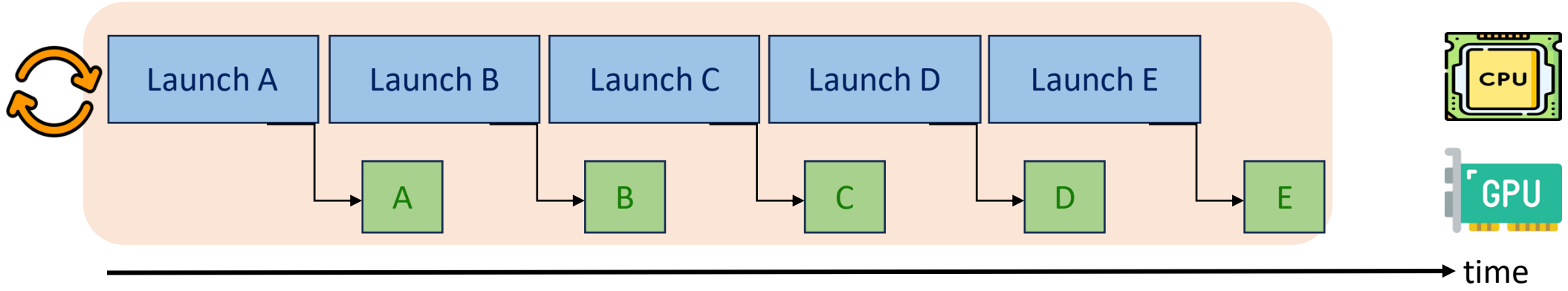


For newer GPUs, kernel execution is fast - CPU launch is bottleneck

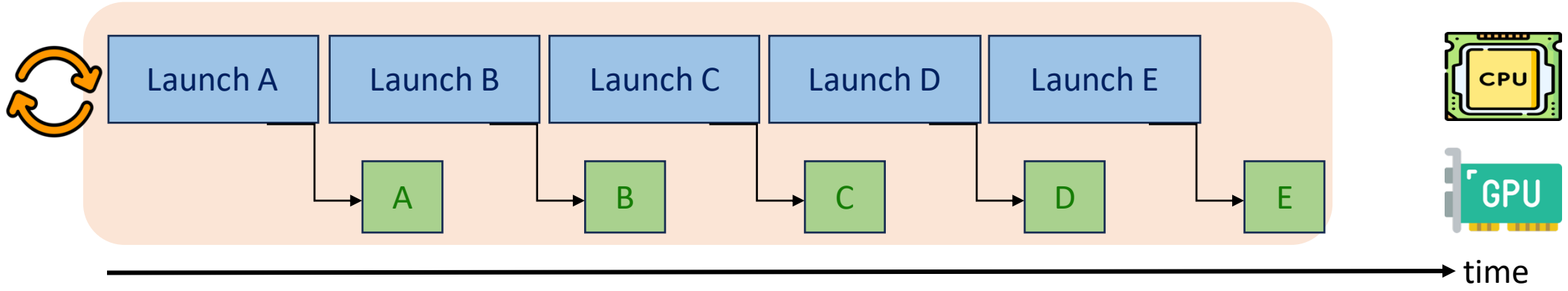
CUDA Graphs Can Reduce Launch Overhead



CUDA Graphs Can Reduce Launch Overhead

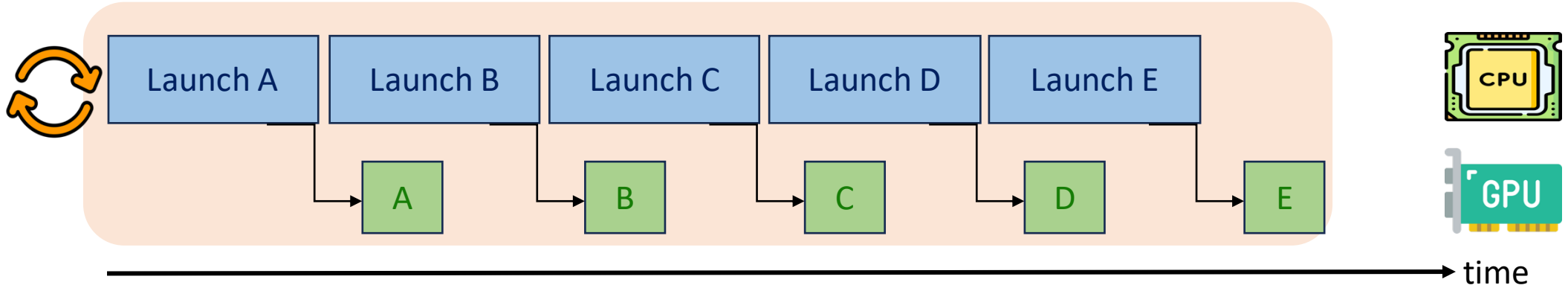


CUDA Graphs Can Reduce Launch Overhead



Capture Graph

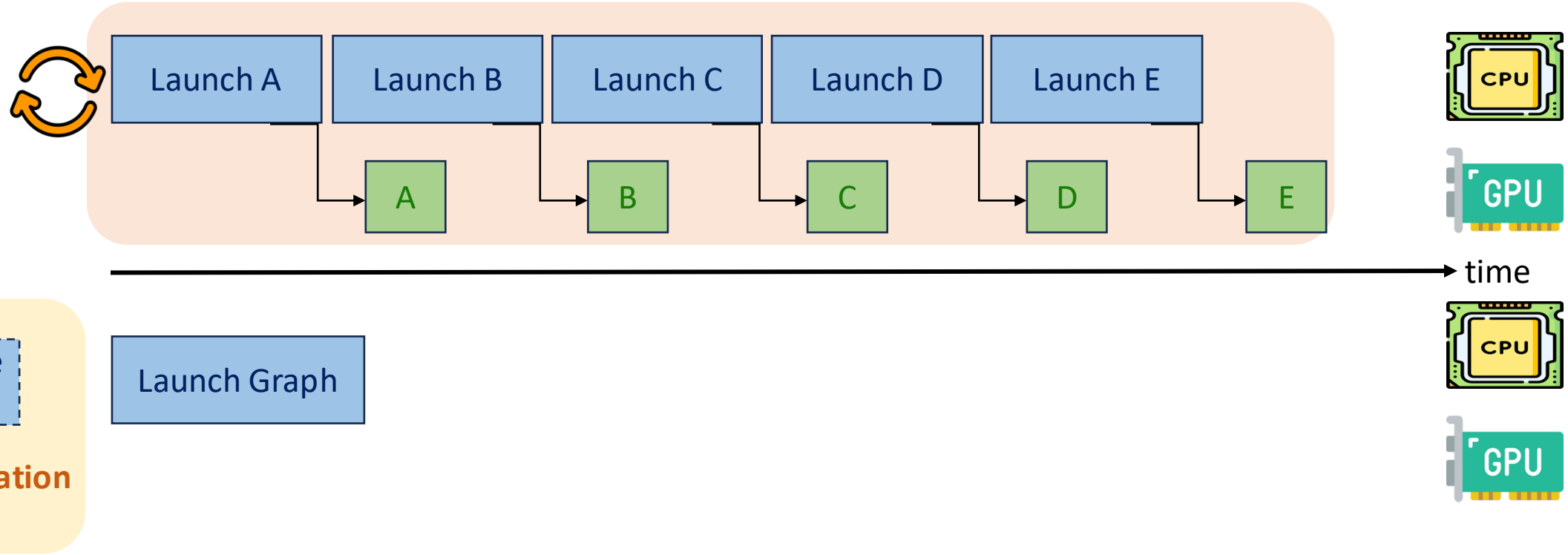
CUDA Graphs Can Reduce Launch Overhead



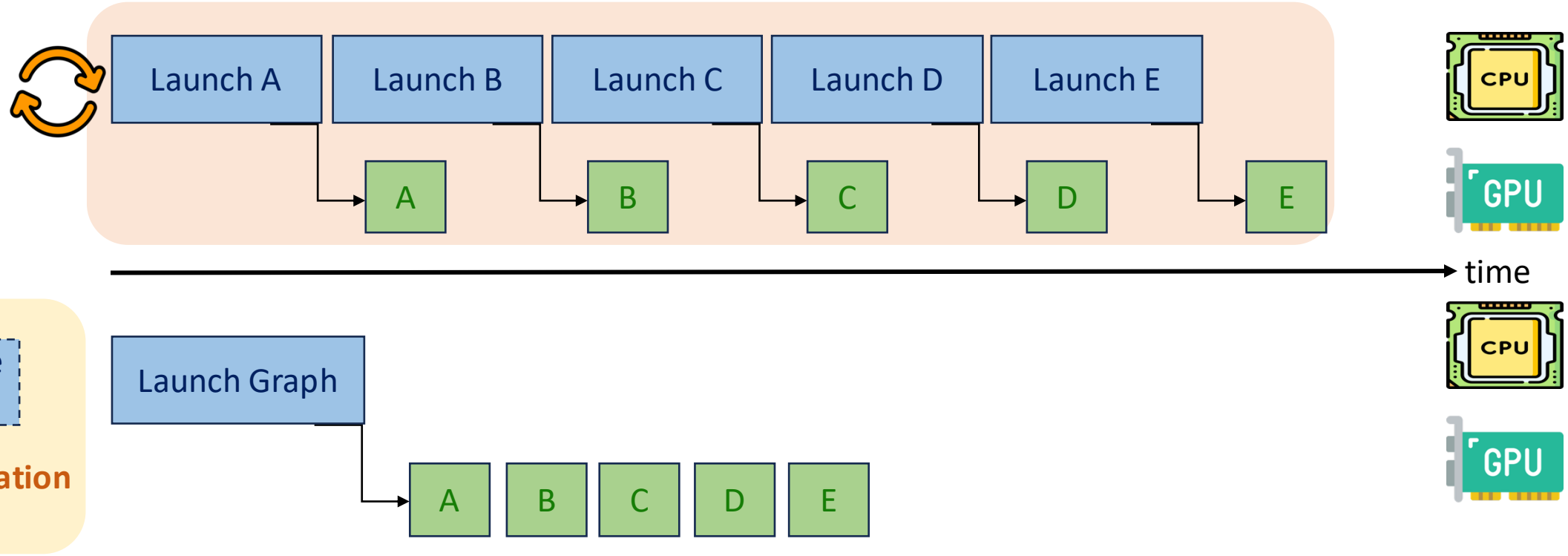
Capture Graph

Slow compilation phase

CUDA Graphs Can Reduce Launch Overhead

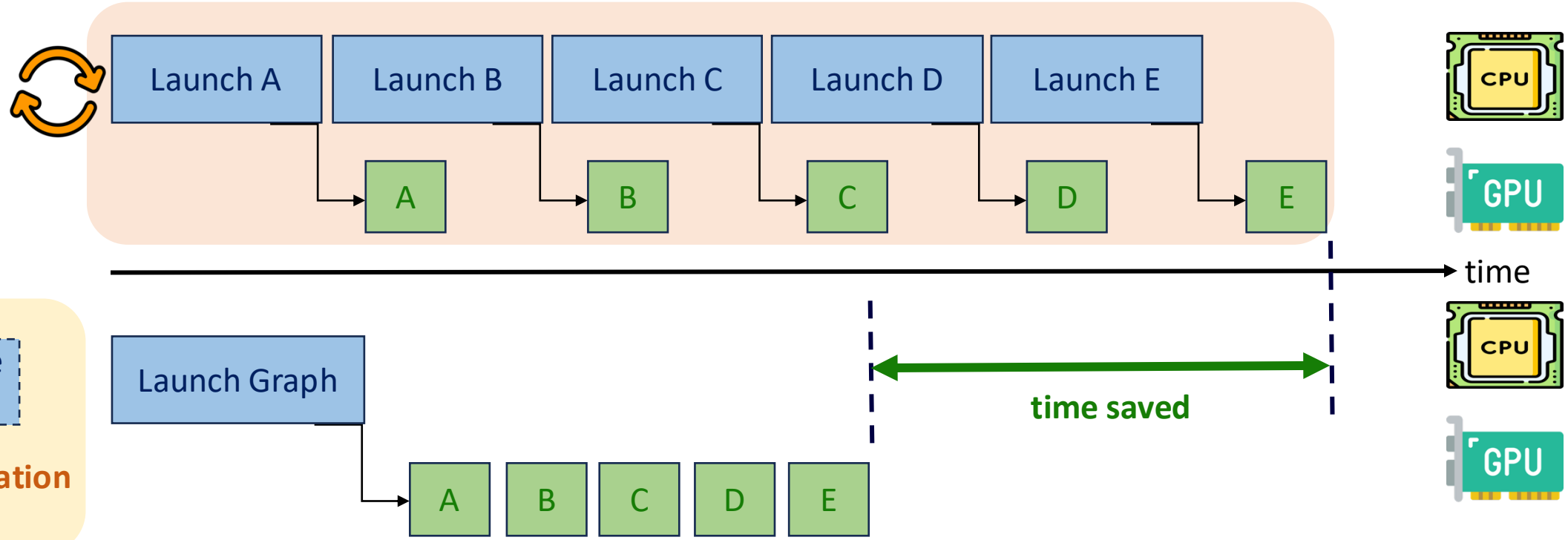


CUDA Graphs Can Reduce Launch Overhead



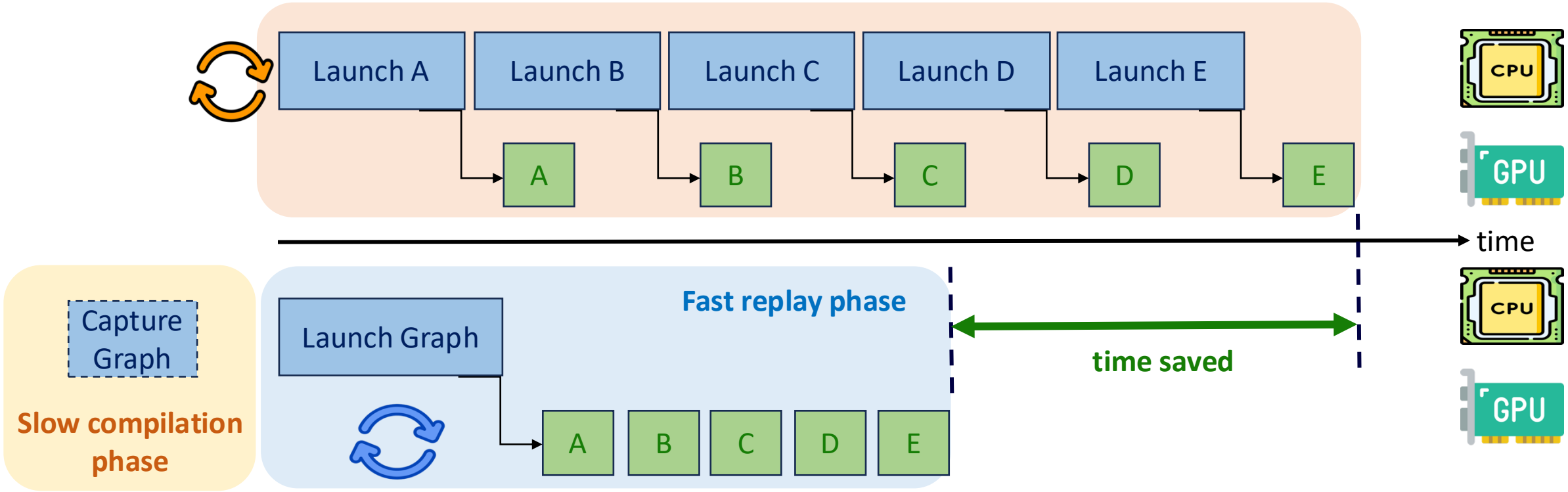
Slow compilation phase

CUDA Graphs Can Reduce Launch Overhead

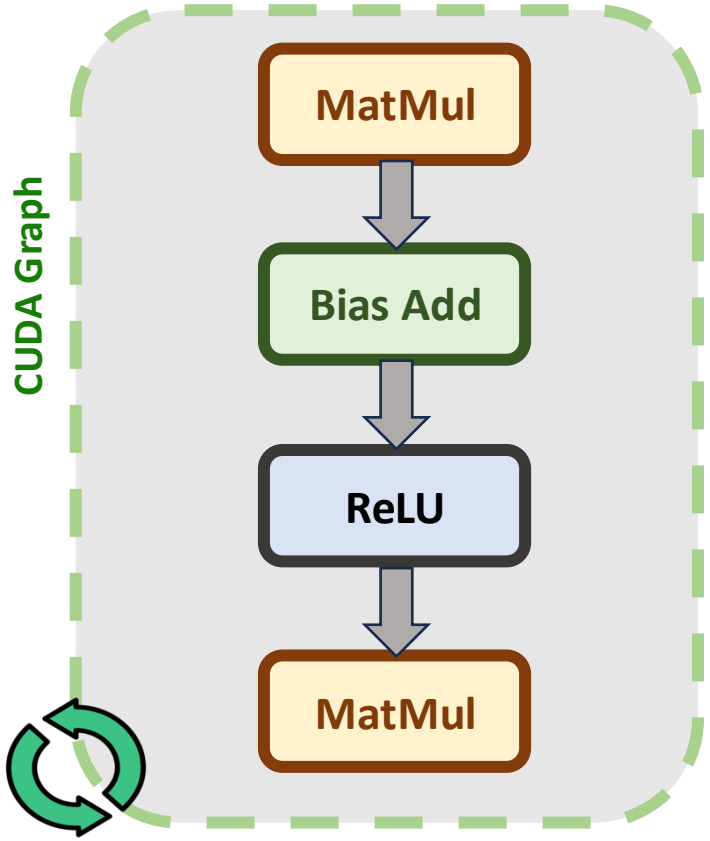


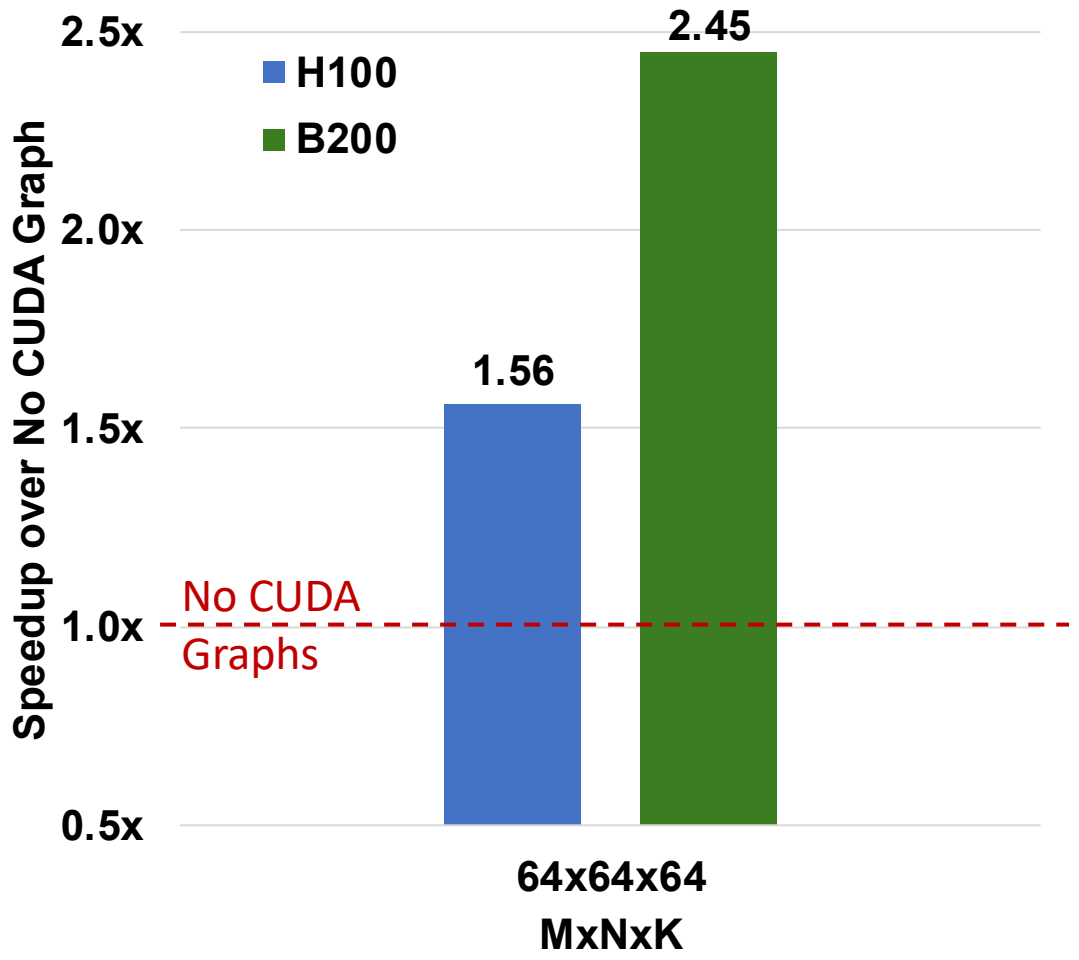
Slow compilation phase

CUDA Graphs Can Reduce Launch Overhead



CUDA Graphs Matter for Performance





Challenges in Deploying CUDA Graphs

1. **Computing on “different” inputs with "static" CUDA Graphs**

CUDA Graphs are Static

Kernel arguments are recorded by value

```
__global__ void kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```

CUDA Graphs are Static

Kernel arguments are recorded by value

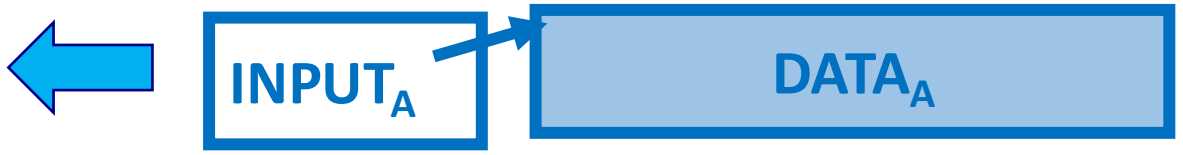
```
__global__ void kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```



CUDA Graphs are Static

Kernel arguments are recorded by value

```
__global__ void kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```



CUDA Graphs are Static

Kernel arguments are recorded by value

```
__global__ void kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```





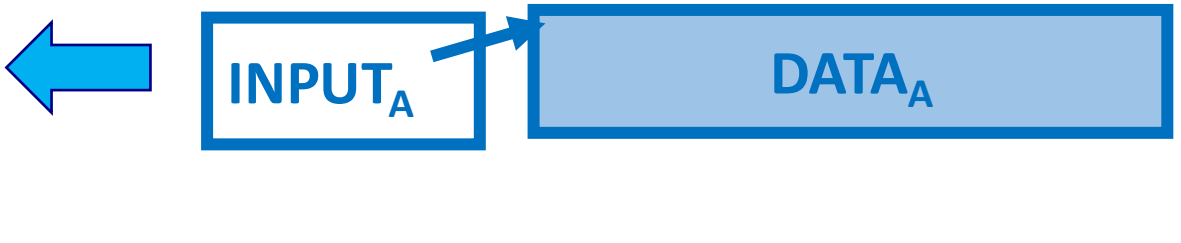


RECORDED BY VALUE

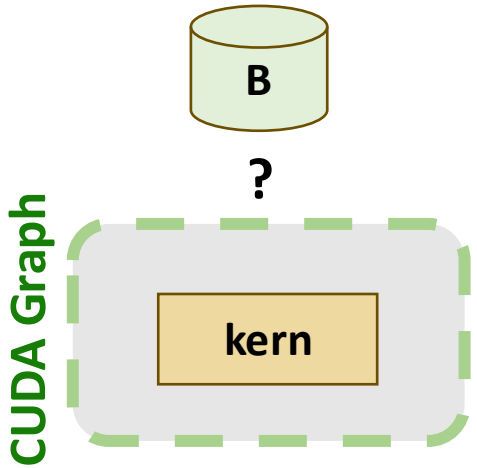
CUDA Graphs are Static

Kernel arguments are recorded by value

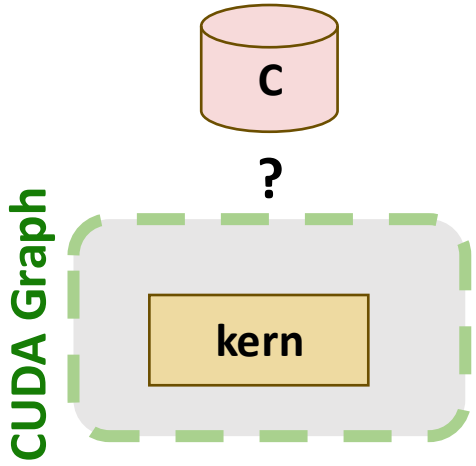
```
__global__ void kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```



RECORDED BY VALUE



Replay-1

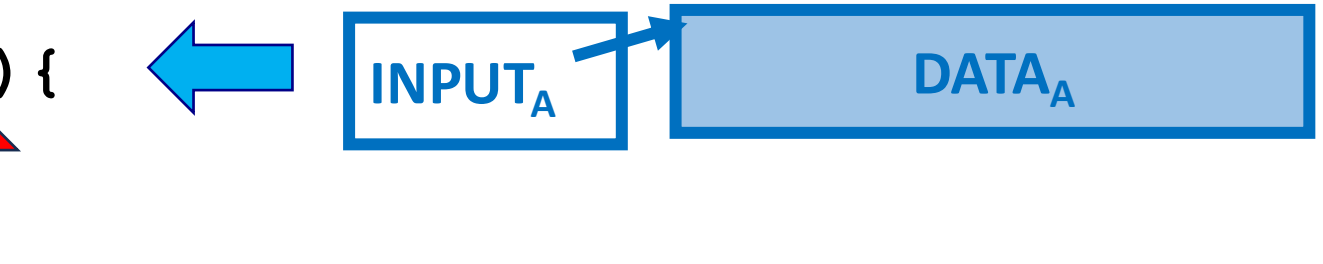


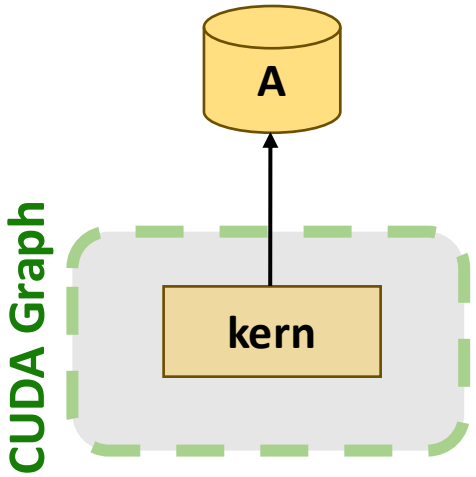
Replay-2

CUDA Graphs are Static

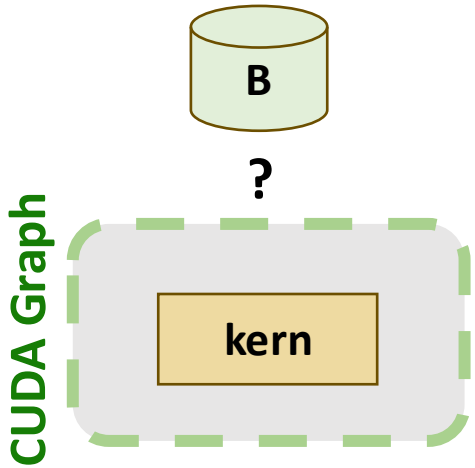
Kernel arguments are recorded by value

```
__global__ void kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```

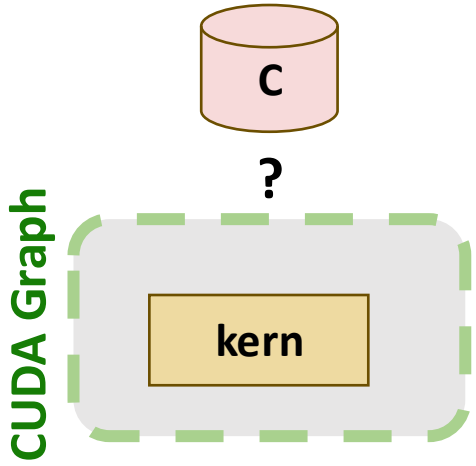




Capture



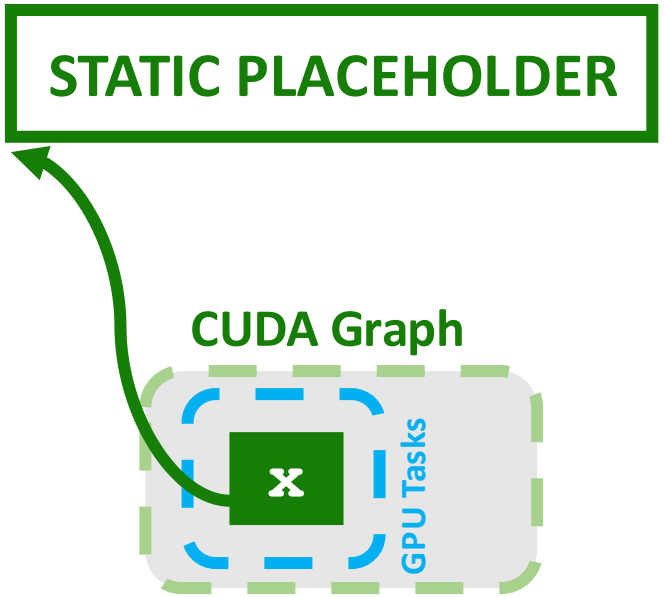
Replay-1



Replay-2

Data Copy on Every Replay

Data copied into a static placeholder on critical path



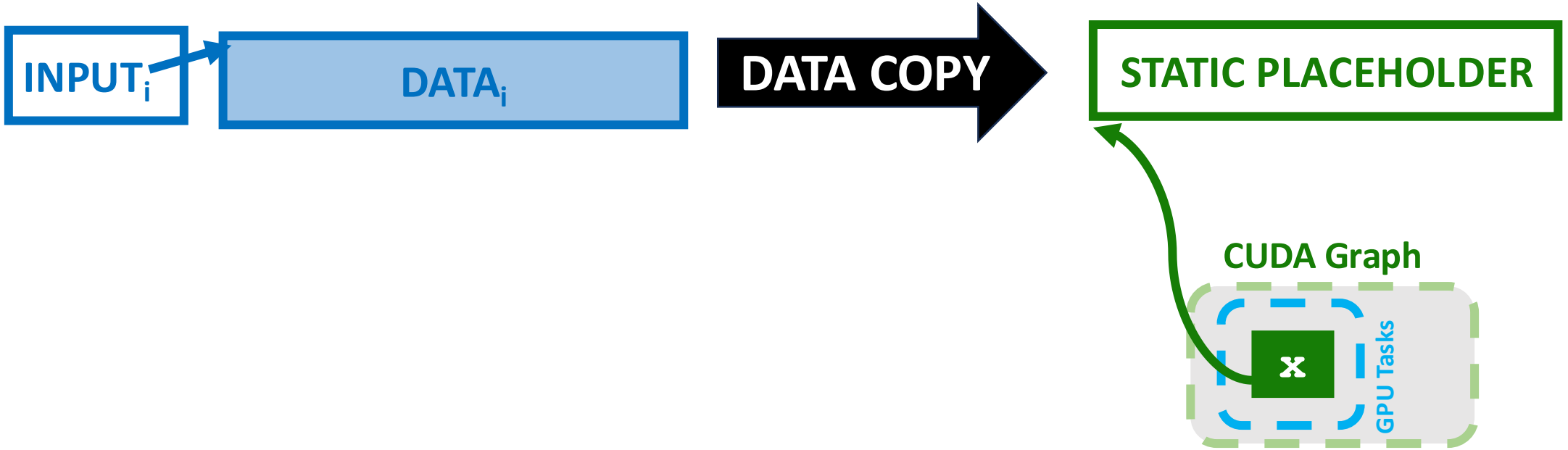
Data Copy on Every Replay

Data copied into a static placeholder on critical path



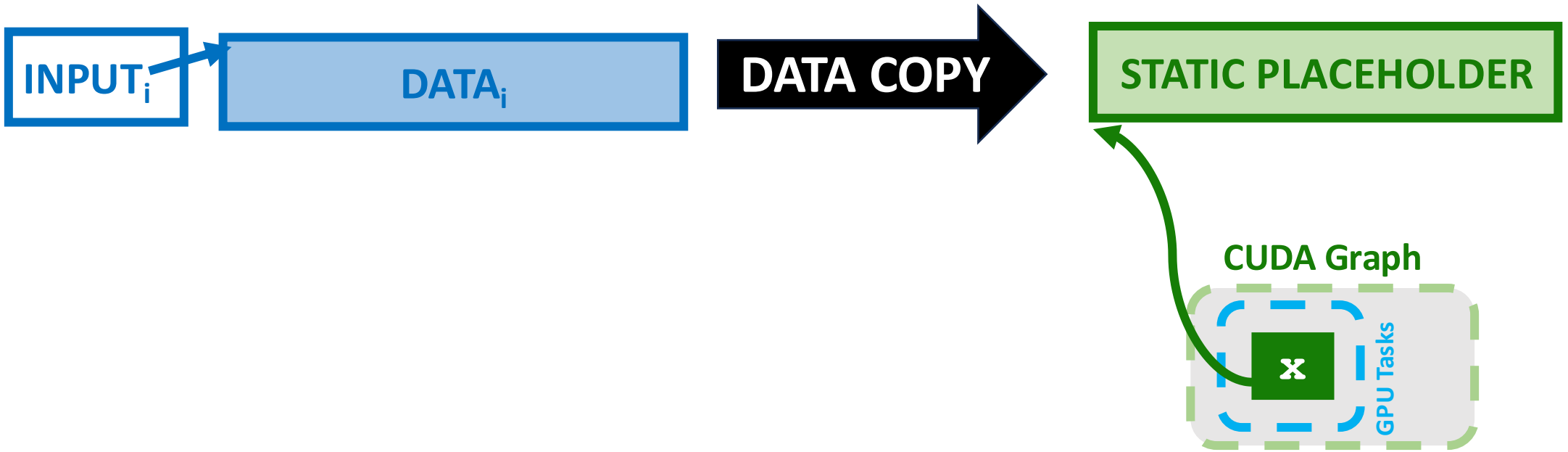
Data Copy on Every Replay

Data copied into a static placeholder on critical path



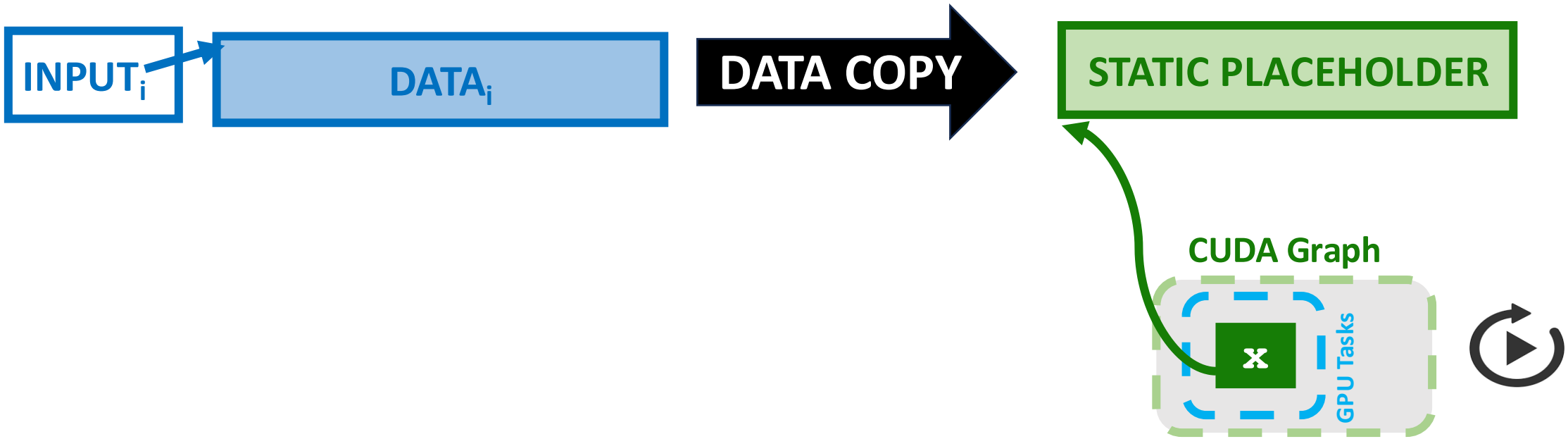
Data Copy on Every Replay

Data copied into a static placeholder on critical path



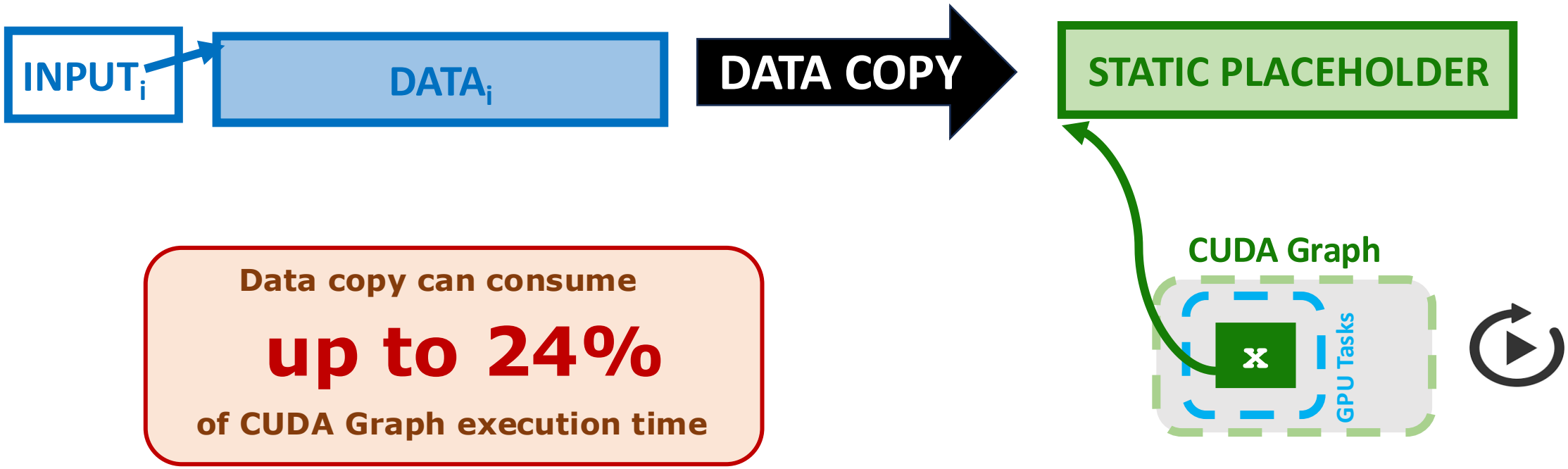
Data Copy on Every Replay

Data copied into a static placeholder on critical path



Data Copy on Every Replay

Data copied into a static placeholder on critical path



Data copy can consume

up to 24%

of CUDA Graph execution time



GraCE: Converting Data Copy to Pointer Copy



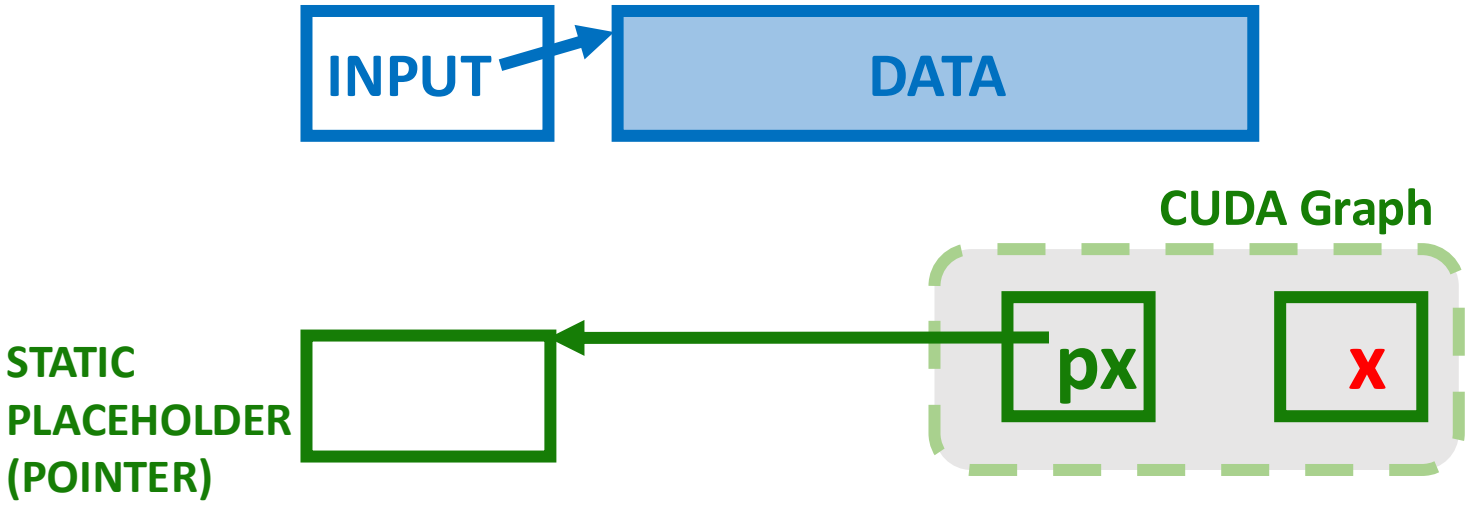
GraCE: Converting Data Copy to Pointer Copy

```
__global__ void kern(int **px){
  int *x = *px; // pointer deref.
  x[idx] = x[idx] + 1;
  ...
}
```



GraCE: Converting Data Copy to Pointer Copy

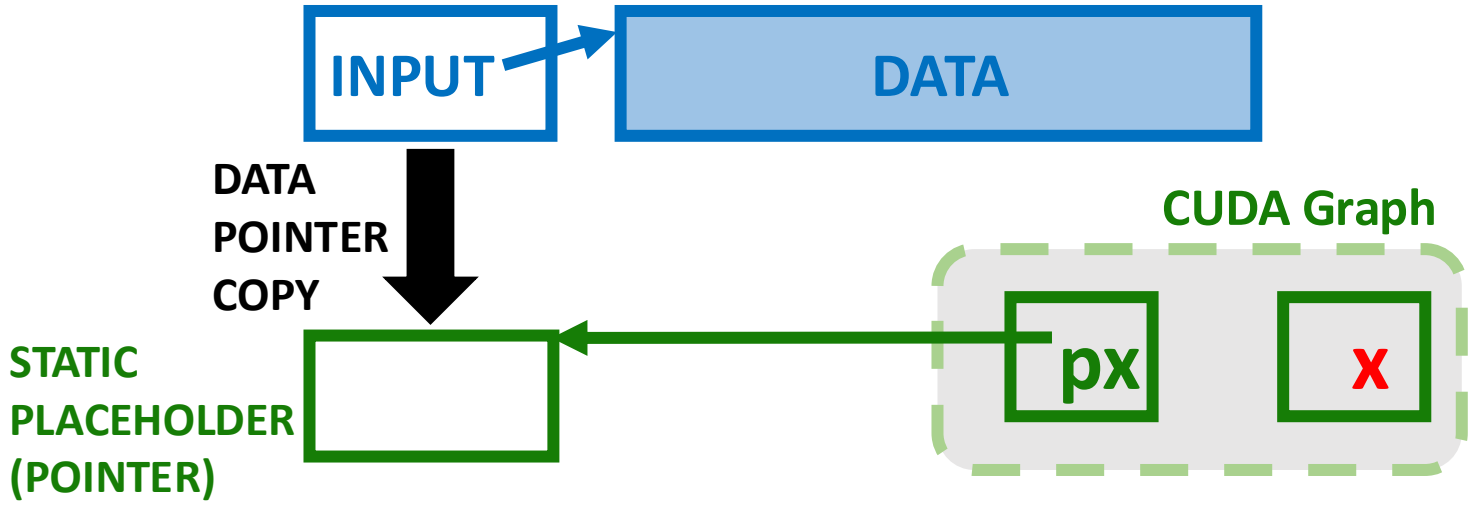
```
__global__ void kern(int **px){
  int *x = *px; // pointer deref.
  x[idx] = x[idx] + 1;
  ...
}
```





GraCE: Converting Data Copy to Pointer Copy

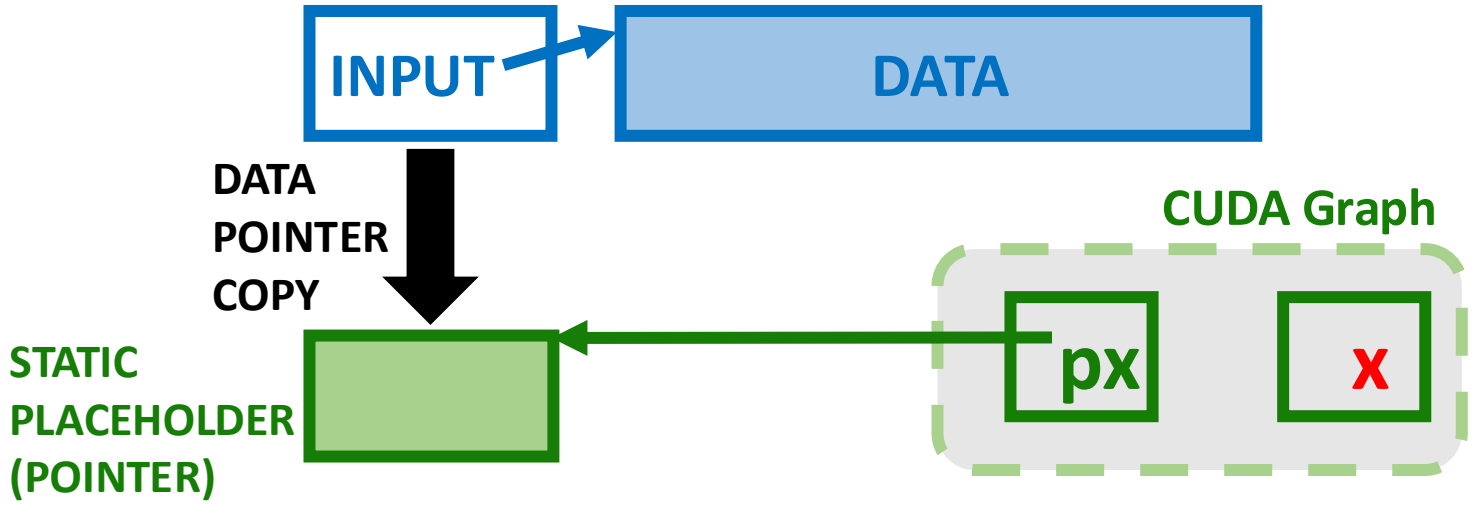
```
__global__ void kern(int **px){
  int *x = *px; // pointer deref.
  x[idx] = x[idx] + 1;
  ...
}
```





GraCE: Converting Data Copy to Pointer Copy

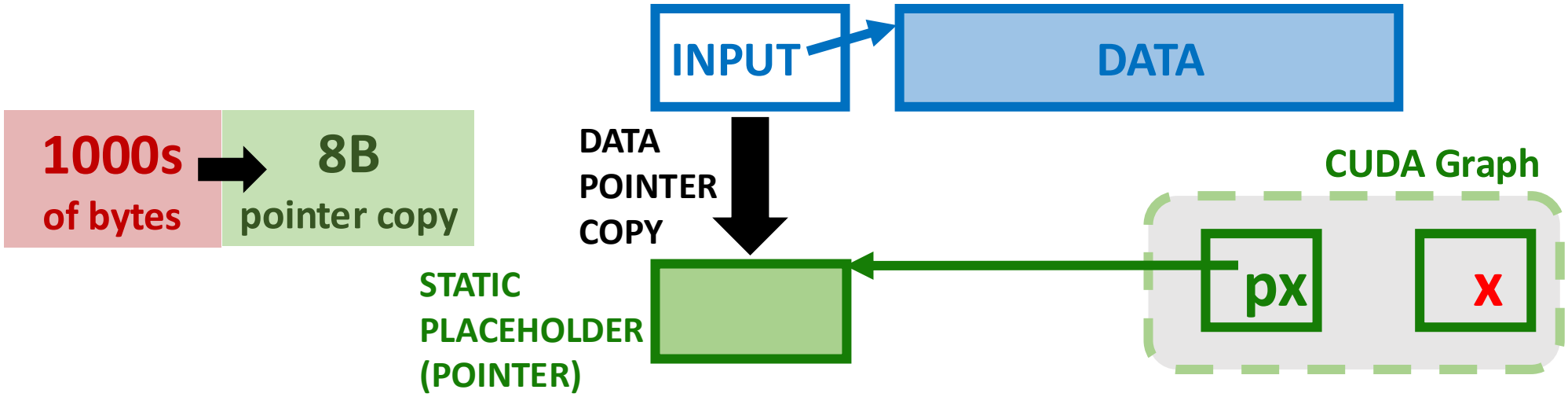
```
__global__ void kern(int **px){
  int *x = *px; // pointer deref.
  x[idx] = x[idx] + 1;
  ...
}
```





GraCE: Converting Data Copy to Pointer Copy

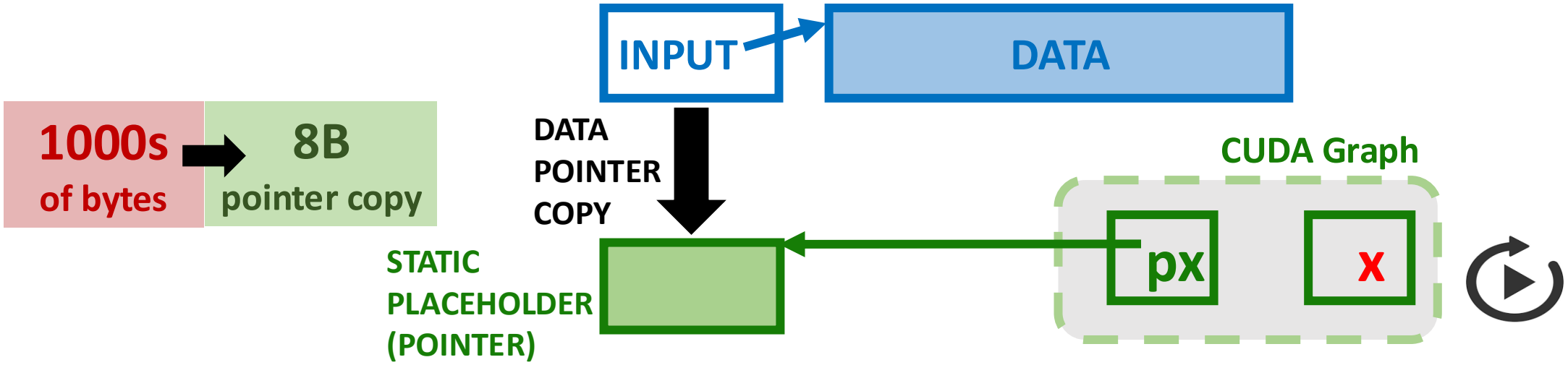
```
__global__ void kern(int **px){
  int *x = *px; // pointer deref.
  x[idx] = x[idx] + 1;
  ...
}
```





GraCE: Converting Data Copy to Pointer Copy

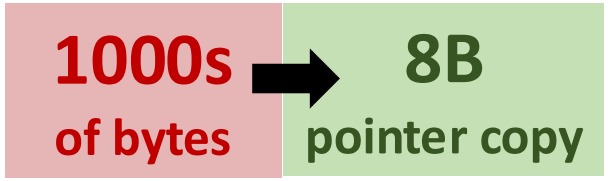
```
__global__ void kern(int **px){
  int *x = *px; // pointer deref.
  x[idx] = x[idx] + 1;
  ...
}
```

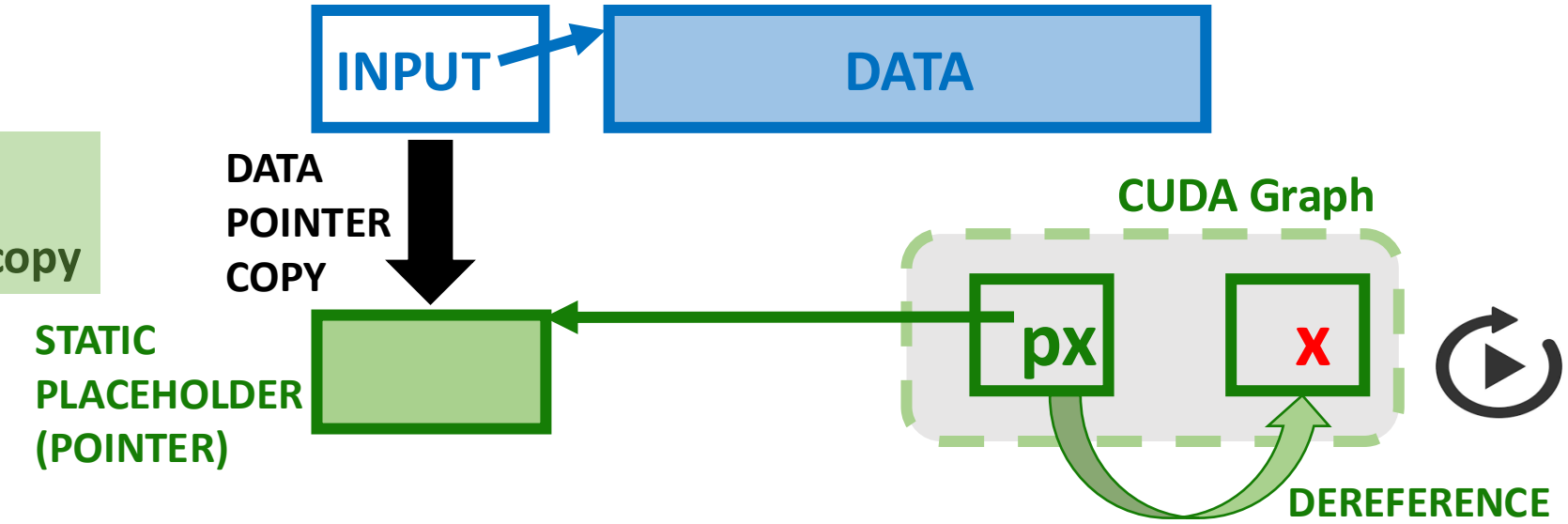




GraCE: Converting Data Copy to Pointer Copy

```
__global__ void kern(int **px){
  int *x = *px; // pointer deref.
  x[idx] = x[idx] + 1;
  ...
}
```



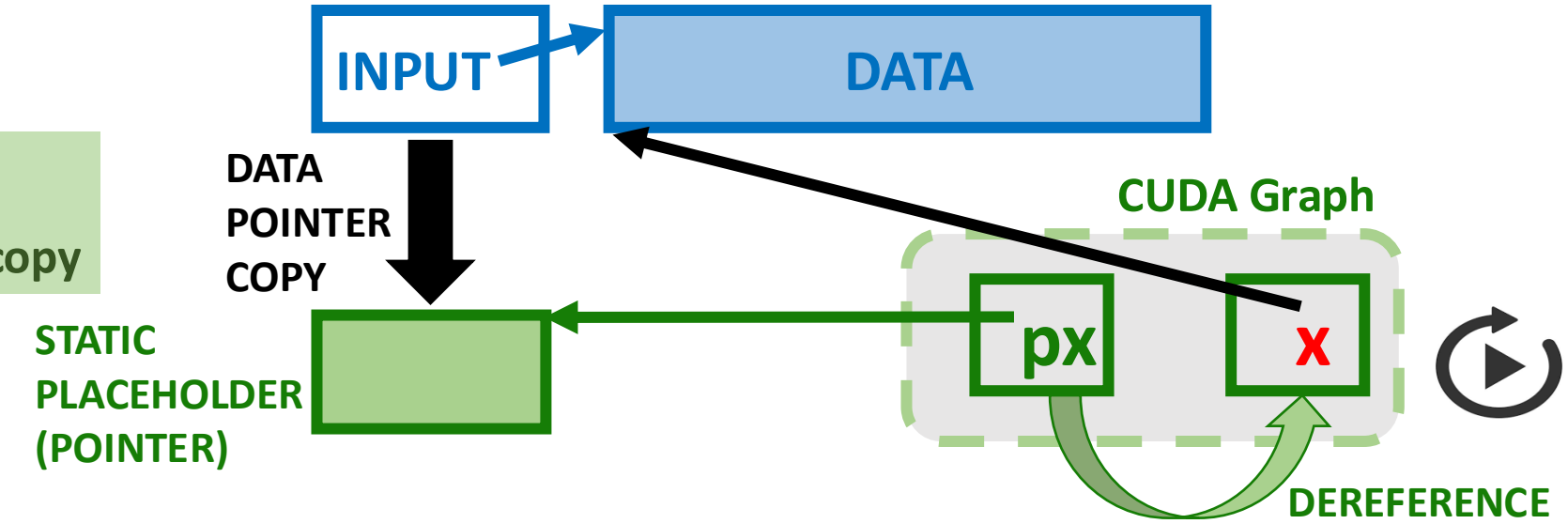




GraCE: Converting Data Copy to Pointer Copy

```
__global__ void kern(int **px){
  int *x = *px; // pointer deref.
  x[idx] = x[idx] + 1;
  ...
}
```

1000s of bytes → 8B pointer copy



Two Kernel Sources, Both Matter

PyTorch2 compiler has a mix of JIT-generated and pre-compiled kernels

56%* **JIT-generated (Triton)**

44%* **Pre-compiled**

JIT-generated kernels

Triton kernels are generated JIT during compilation. GraCE can modify them on the fly

Easy

Pre-compiled kernels

For pre-compiled kernels like cuBLAS, cuDNN, GraCE ***cannnot*** modify kernel code on the fly

Hard

** Based on workloads from TorchBench, TIMM, Huggingface*

Two Kernel Sources, Both Matter

PyTorch2 compiler has a mix of JIT-generated and pre-compiled kernels

56%* **JIT-generated (Triton)**

44%* **Pre-compiled**

JIT-generated kernels

Triton kernels are generated JIT during compilation. GraCE can modify them on the fly

Easy

Pre-compiled kernels

For pre-compiled kernels like cuBLAS, cuDNN, GraCE ***cannnot*** modify kernel code on the fly

Hard

** Based on workloads from TorchBench, TIMM, Huggingface*



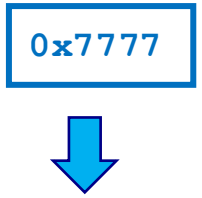
GraCE: Handling Pre-compiled Kernels

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```



GraCE: Handling Pre-compiled Kernels

Dummy known pointer




```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```



GraCE: Handling Pre-compiled Kernels

Dummy known pointer

0x7777

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```

CAPTURE GRAPH



GraCE: Handling Pre-compiled Kernels

Dummy known pointer

0x7777

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```

CAPTURE GRAPH

CUDA Graph

precompiled_kern

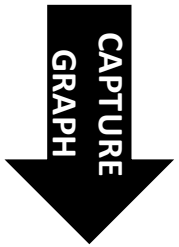


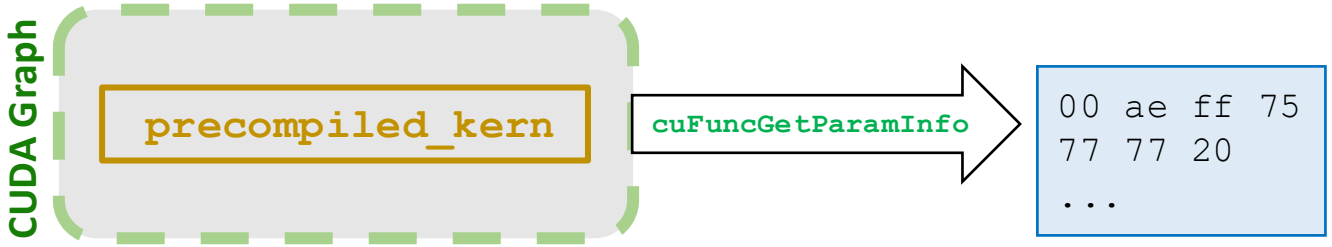
GraCE: Handling Pre-compiled Kernels

Dummy known pointer




```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```





Kernel's Param Buffer



GraCE: Handling Pre-compiled Kernels

Dummy known pointer `0x7777`

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```

CAPTURE GRAPH

CUDA Graph: `precompiled_kern`

`cuFuncGetParamInfo`

Dummy known pointer `0x7777`

```
00 ae ff 75
77 77 20
...
```

Byte Pattern Match

Kernel's Param Buffer



GraCE: Handling Pre-compiled Kernels

Dummy known pointer: 0x7777

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```

CAPTURE GRAPH

CUDA Graph: precompiled_kern

cuFuncGetParamInfo

Dummy known pointer: 0x7777

offset

00 ae ff 75
77 77 20
...

Byte Pattern Match

Kernel's Param Buffer



GraCE: Handling Pre-compiled Kernels

Dummy known pointer: 0x7777

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```

CAPTURE GRAPH

CUDA Graph: precompiled_kern

cuFuncGetParamInfo

Dummy known pointer: 0x7777

offset

00 ae ff 75
77 77 20
...

Byte Pattern Match

Kernel's Param Buffer

```
__global__ void
prelude_kern(int **px){
  cudaGraphKernelNodeSetParam(px);
  ...
}
```

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```



GraCE: Handling Pre-compiled Kernels

Dummy known pointer: 0x7777

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```

CAPTURE GRAPH

CUDA Graph

```
__global__ void
prelude_kern(int **px){
  cudaGraphKernelNodeSetParam(px);
  ...
}
```

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```

CUDA Graph

precompiled_kern

cuFuncGetParamInfo

Dummy known pointer: 0x7777

offset

00 ae ff 75
77 77 20
...

Kernel's Param Buffer

Byte Pattern Match



GraCE: Handling Pre-compiled Kernels

Dummy known pointer: `0x7777`

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```

CAPTURE GRAPH

CUDA Graph

```
precompiled_kern
```

cuFuncGetParamInfo

Dummy known pointer: `0x7777`

offset

```
00 ae ff 75
77 77 20
...
```

Byte Pattern Match

Kernel's Param Buffer

CUDA Graph

```
__global__ void
prelude_kern(int **px){
  cudaGraphKernelNodeSetParam(px);
  ...
}
```

```
int *x = *px;
```

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```



GraCE: Handling Pre-compiled Kernels

Dummy known pointer: 0x7777

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```

CAPTURE GRAPH

CUDA Graph: precompiled_kern

cuFuncGetParamInfo

Dummy known pointer: 0x7777

offset

00 ae ff 75
77 77 20
...

Kernel's Param Buffer

Byte Pattern Match

CUDA Graph

```
__global__ void
prelude_kern(int **px){
  cudaGraphKernelNodeSetParam(px);
  ...
}
```

```
int *x = *px;
```

write

```
__global__ void
precompiled_kern(int *x){
  x[idx] = x[idx] + 1;
  ...
}
```



GraCE: From Data Copy to Byte-Scale Pointer Copy

Applications*	Bytes Copy Reduced Per Replay
Deep Recommender (I)	3.0 GB → 8 B
BlenderBot-CondGen (T)	1.6 GB → 56 B
MT5-CondGen (T)	1.0 GB → 312 B
DistilGPT2 (T)	953 MB → 136 B
Speech2Text2-CLM (T)	849 MB → 136 B
DeBERTa-MLM (T)	549 MB → 136 B

**workloads from TorchBench, TIMM, Huggingface*

I/T: Inference/Training

GraCE reduces replay-time copied data by more than 99% in workloads having large copies

Challenges in Deploying CUDA Graphs

1. Computing on “different” inputs with "static" CUDA Graphs

2. **Many program segments are CUDA Graph-unfriendly**



GraCE: Increasing Coverage with Code Transformation

Speech Transformer

```
...

temp = np.power(d_k, 0.5)



self.attn = ScaledDotProductAttention(
     temperature=temp
)
...
attn = torch.bmm(q, k.transpose(1, 2))

attn = attn / self.temperature

...
```



GraCE: Increasing Coverage with Code Transformation

Speech Transformer

```
...

temp = np.power(d_k, 0.5)



self.attn = ScaledDotProductAttention(
     temperature=temp
)
...
attn = torch.bmm(q, k.transpose(1, 2))

attn = attn / self.temperature

...
```



GraCE: Increasing Coverage with Code Transformation

Speech Transformer

```
...

temp = np.power(d_k, 0.5)



self.attn = ScaledDotProductAttention(
     temperature=temp
)
...
attn = torch.bmm(q, k.transpose(1, 2))

attn = attn / self.temperature

...
```



GraCE: Increasing Coverage with Code Transformation

Speech Transformer

```
...

temp = np.power(d_k, 0.5)



self.attn = ScaledDotProductAttention(
     temperature=temp
)
...
attn = torch.bmm(q, k.transpose(1, 2))

attn = attn / self.temperature

...
```





GraCE: Increasing Coverage with Code Transformation

Speech Transformer

```
...

temp = np.power(d_k, 0.5)



self.attn = ScaledDotProductAttention(
     temperature=temp
)
...
attn = torch.bmm(q, k.transpose(1, 2))

attn = attn / self.temperature

...
```



Cannot Capture



GraCE: Increasing Coverage with Code Transformation

Speech Transformer

```
...

temp = np.power(d_k, 0.5)



self.attn = ScaledDotProductAttention(
     temperature=temp
)
...
attn = torch.bmm(q, k.transpose(1, 2))

attn = attn / self.temperature

...
```

Cannot Capture

GraCE

```
...

temp = torch.tensor(
     np.power(d_k, 0.5),
     device='cuda'
)

self.attn = ScaledDotProductAttention(
     temperature=temp
)
...
attn = torch.bmm(q, k.transpose(1, 2))

attn = attn / self.temperature

...
```



GraCE: Increasing Coverage with Code Transformation

Speech Transformer

```
...
temp = np.power(d_k, 0.5)

self.attn = ScaledDotProductAttention(
     temperature=temp
)
...
attn = torch.bmm(q, k.transpose(1, 2))
attn = attn / self.temperature
...
```

Cannot Capture

GraCE

```
...
temp = torch.tensor(
     np.power(d_k, 0.5),
     device='cuda'
)

self.attn = ScaledDotProductAttention(
     temperature=temp
)
...
attn = torch.bmm(q, k.transpose(1, 2))
attn = attn / self.temperature
...
```



GraCE: Increasing Coverage with Code Transformation

Speech Transformer

```
...
temp = np.power(d_k, 0.5)

self.attn = ScaledDotProductAttention(
     temperature=temp
)
...
attn = torch.bmm(q, k.transpose(1, 2))
attn = attn / self.temperature
...
```

Cannot Capture

GraCE

CUDA Graph

```
...
temp = torch.tensor(
     np.power(d_k, 0.5),
     device='cuda'
)

self.attn = ScaledDotProductAttention(
     temperature=temp
)
...
attn = torch.bmm(q, k.transpose(1, 2))
attn = attn / self.temperature
...
```

Can Capture



GraCE: Code Transformations Unlock Large CUDA Graphs

Applications*	Increase in % of Kernels in CUDA Graphs
Multi-modal CLIP (I)	0 % → 99 %
XLNET (I)	0 % → 99 %
XLNET (T)	1.5 % → 98 %
Speech Transformer (I)	5 % → 74 %
DALL-E2 (I)	79 % → 99 %
Mask R-CNN (I)	58 % → 71 %

**workloads from TorchBench, TIMM, Huggingface;* I/T: Inference/Training

GraCE raises CUDA Graph coverage from near-zero to 74–99%

Challenges in Deploying CUDA Graphs

1. Computing on “different” inputs with "static" CUDA Graphs
2. Many program segments are CUDA Graph-unfriendly
3. **Overheads of CUDA Graphs can outweigh benefits**

CUDA Graphs Are NOT Always Beneficial

CUDA Graphs reduce launch cost, but add replay-time overheads

Benefits

- Fewer CPU launches
- Less GPU idle time

Overheads

- Invariant checks for static pointers
- Graph memory reuse / GC

25% of CUDA Graphs in TorchBench* hurt performance!

* pytorch/benchmark: TorchBench is a collection of open source benchmarks used to evaluate PyTorch performance.

CUDA Graphs Are NOT Always Beneficial

CUDA Graphs reduce launch cost, but add replay-time overheads

Benefits

- Fewer CPU launches
- Less GPU idle time

Overheads

- Invariant checks for static pointers
- Graph memory reuse / GC

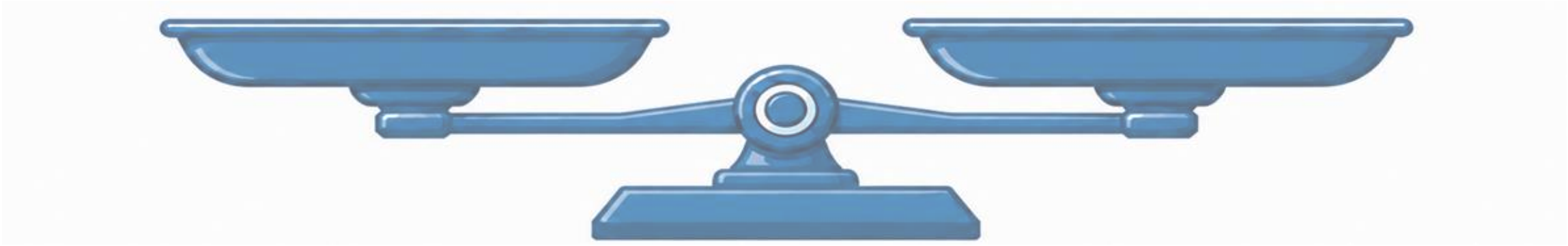
25% of CUDA Graphs in TorchBench* hurt performance!

When the overheads exceed launch savings, CUDA Graphs slow down execution

* pytorch/benchmark: TorchBench is a collection of open source benchmarks used to evaluate PyTorch performance.



GraCE: Selectively Deploy CUDA Graphs

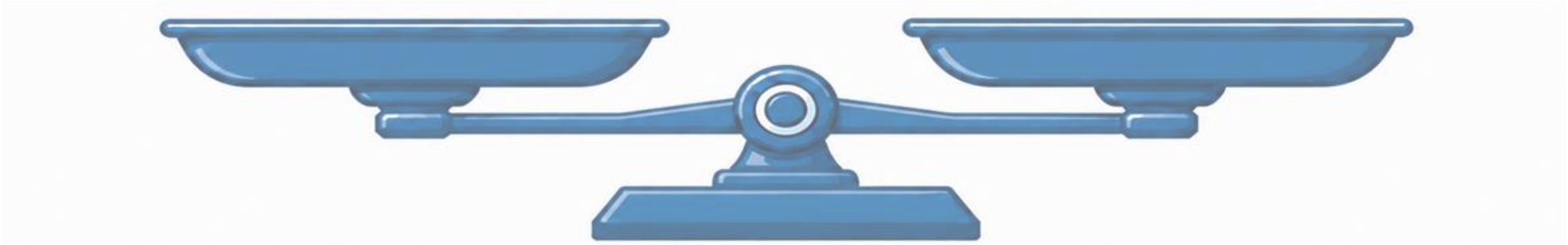




GraCE: Selectively Deploy CUDA Graphs

CUDA Graph Benefits

CUDA Graph Overheads





GraCE: Selectively Deploy CUDA Graphs

CUDA Graph Benefits

CUDA Graph Overheads





GraCE: Selectively Deploy CUDA Graphs

CUDA Graph Benefits

CUDA Graph Overheads



Summary of GraCE's Key Contributions

1. Computing on "different" inputs with "static" CUDA Graphs

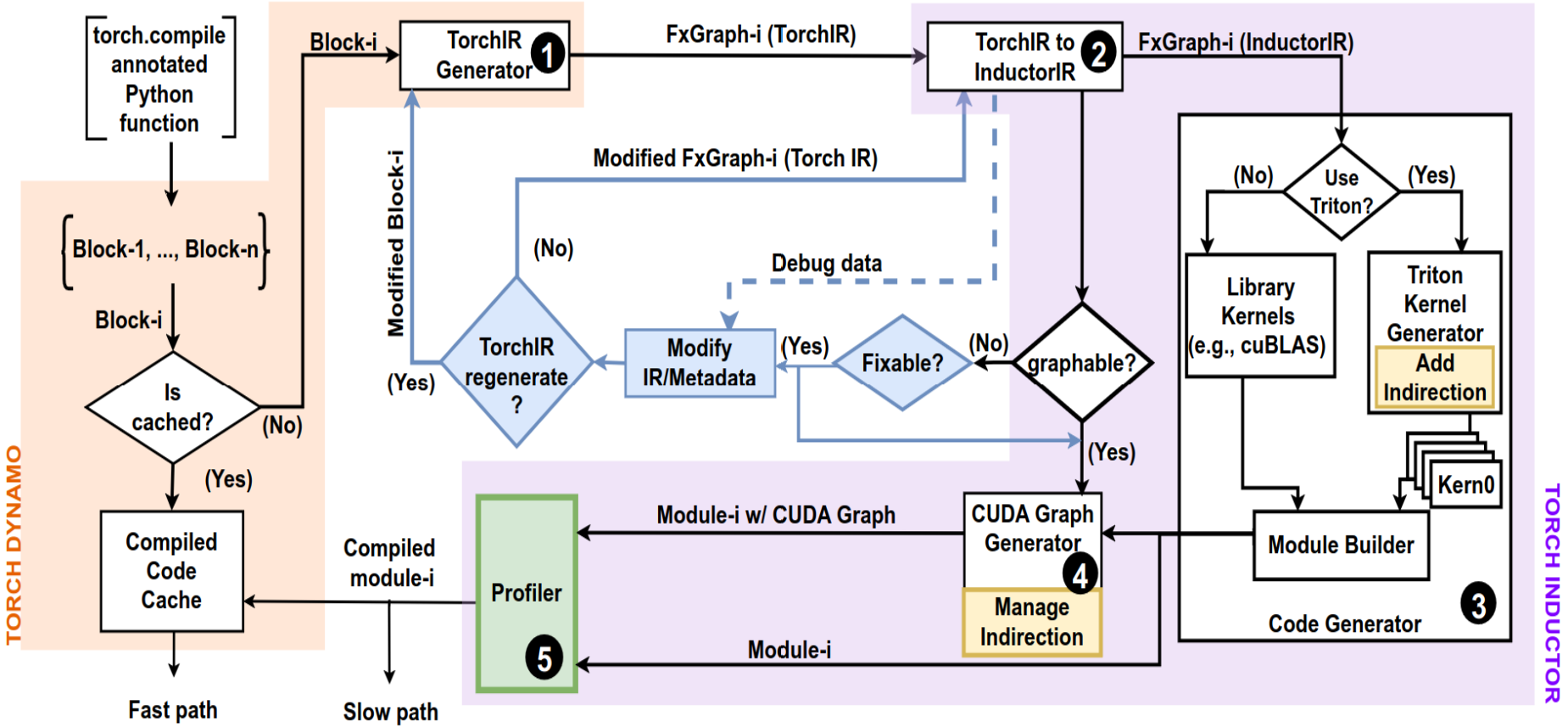
2. Many programs segments are CUDA Graph-unfriendly

3. Overheads of CUDA Graphs can outweigh benefits

Summary of GraCE's Key Contributions

1. Computing on "different" inputs with "static" CUDA Graphs
   - **Parameter Indirection**
2. Many programs segments are CUDA Graph-unfriendly
   - **CUDA Graph-aware Code Transformation**
3. Overheads of CUDA Graphs can outweigh benefits
   - **Selective Deployment of CUDA Graphs**

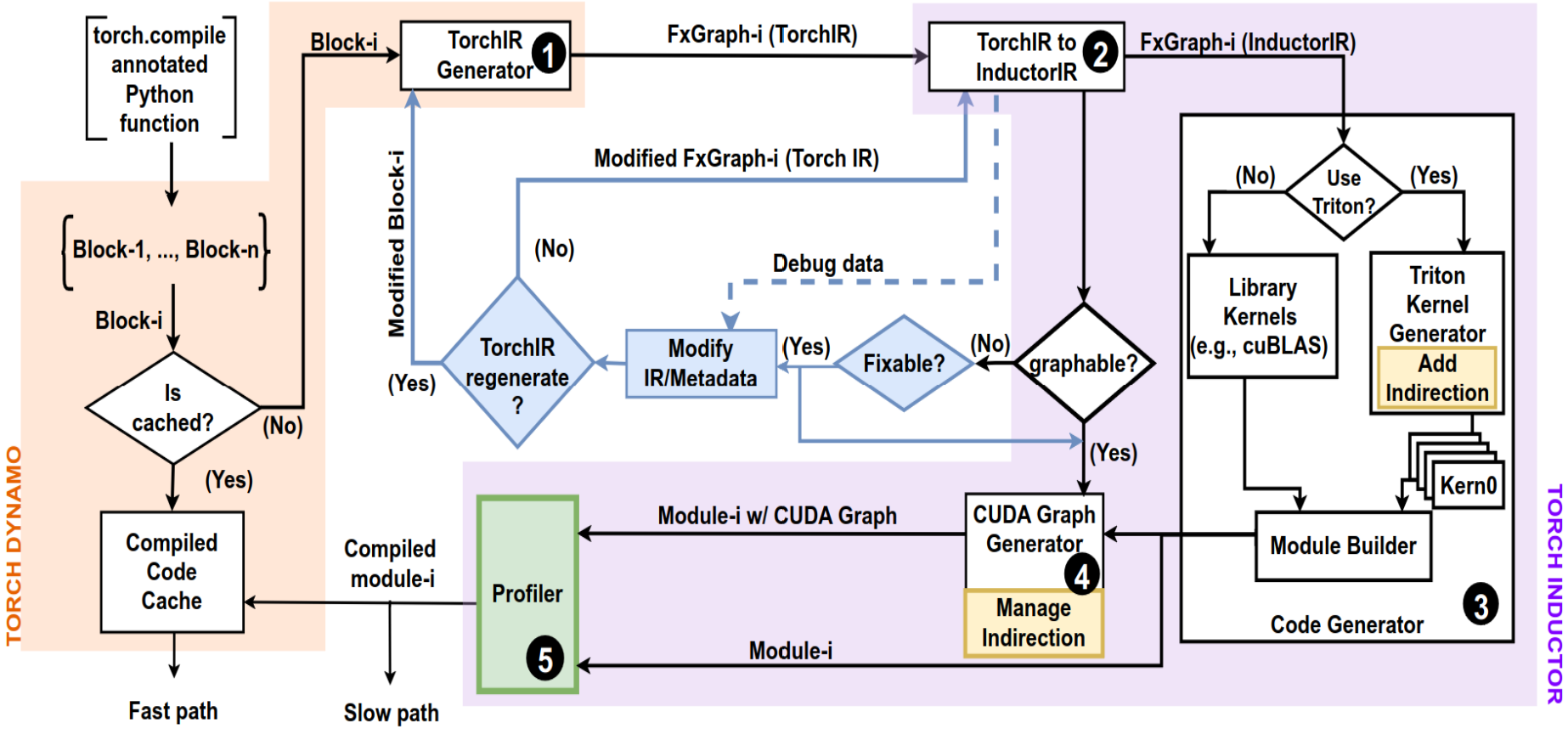
GraCE in PyTorch2 Compiler



No app changes

No hardware changes

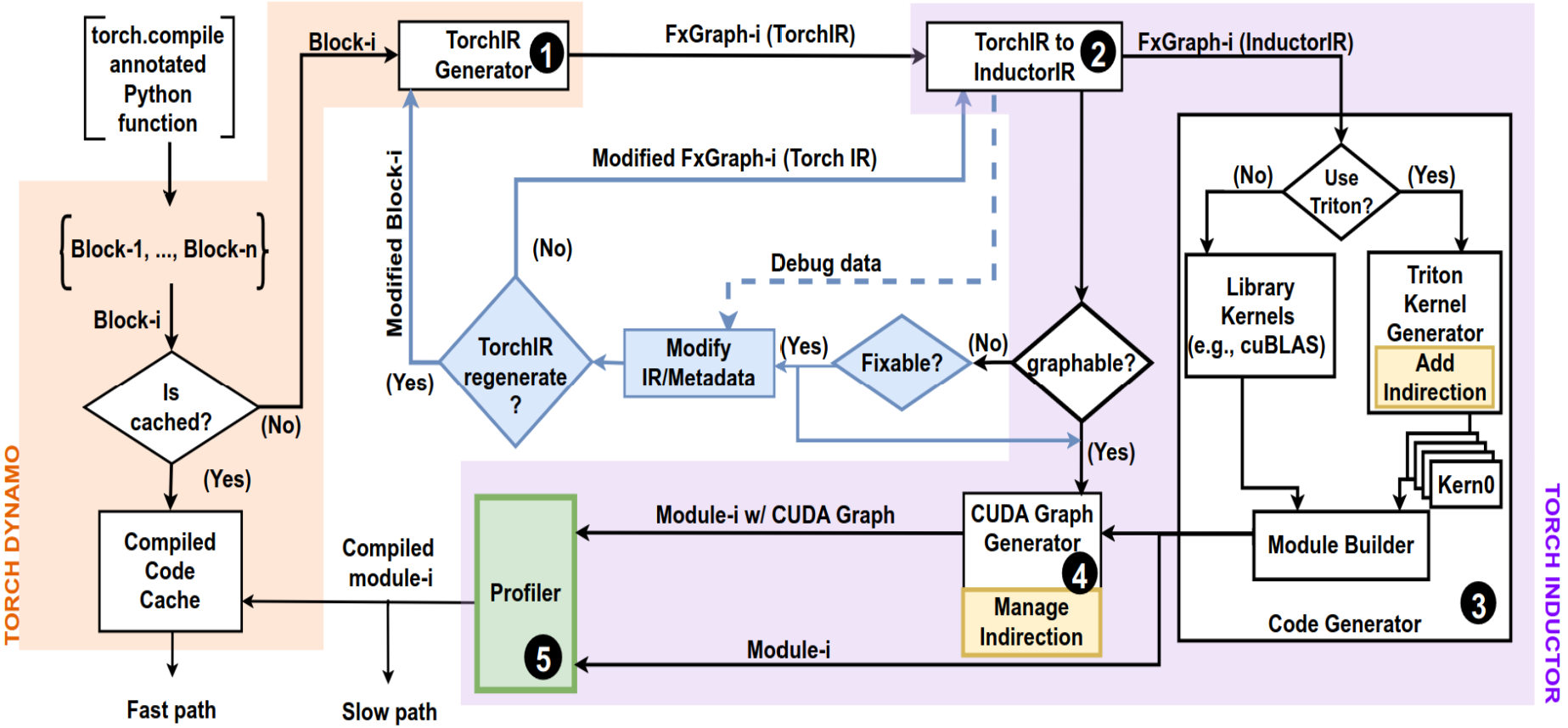
GraCE in PyTorch2 Compiler



No app changes

No hardware changes

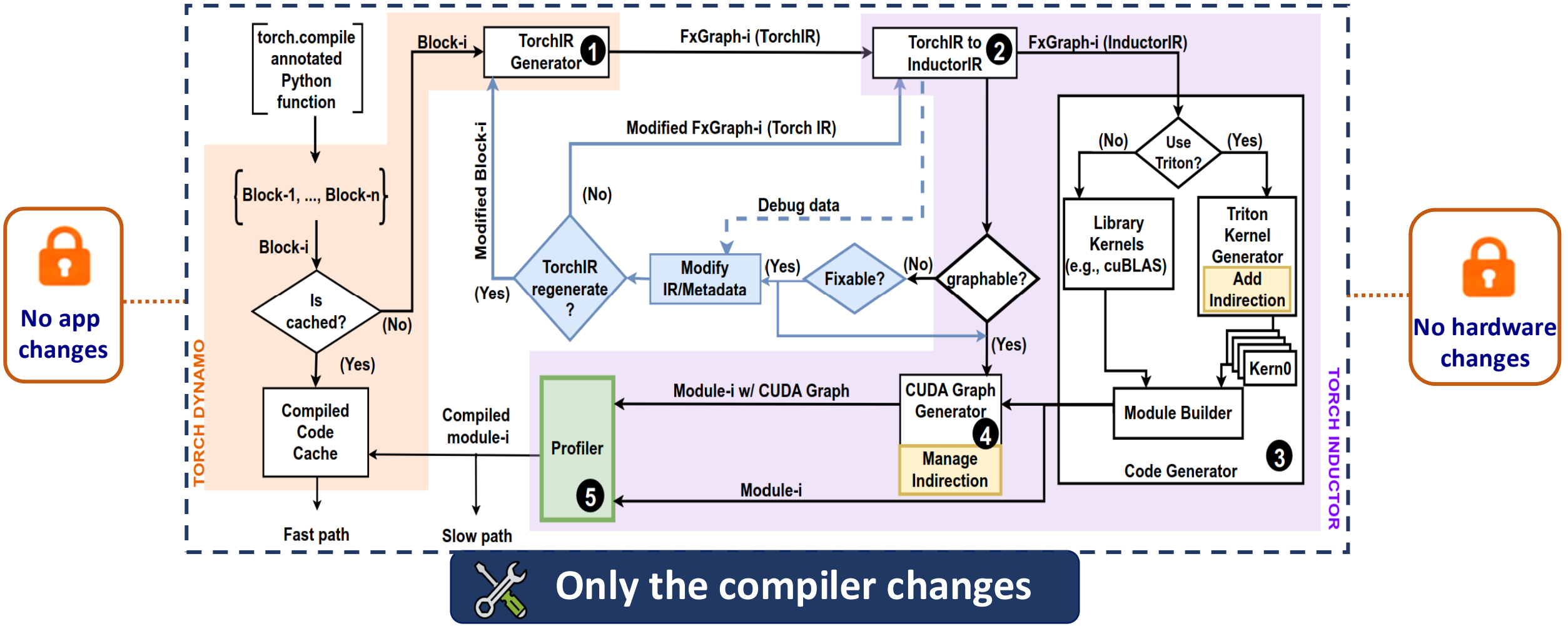
GraCE in PyTorch2 Compiler



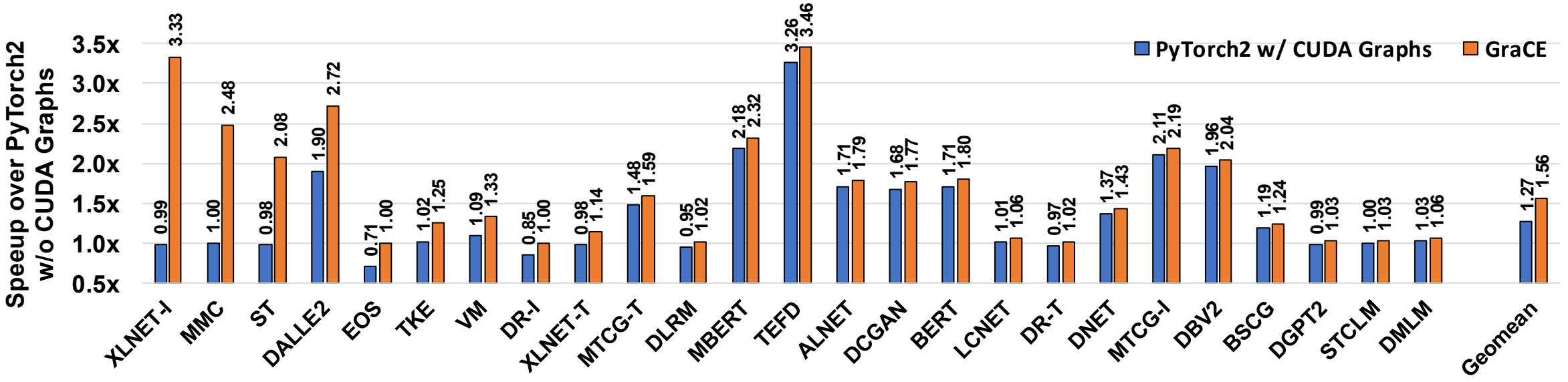
No app changes

No hardware changes

GraCE in PyTorch2 Compiler

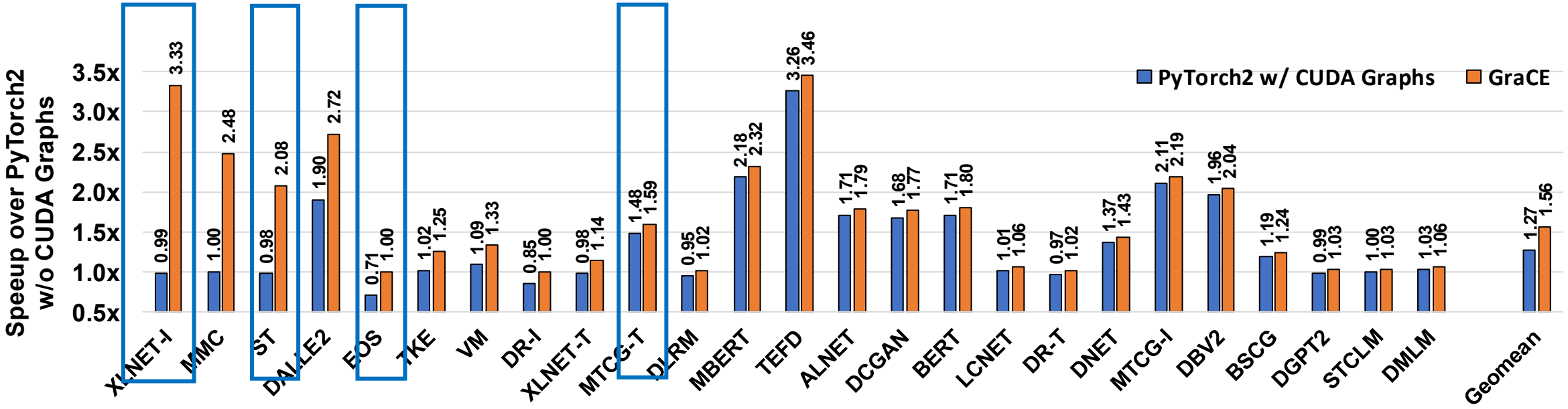


Performance Gains with GraCE



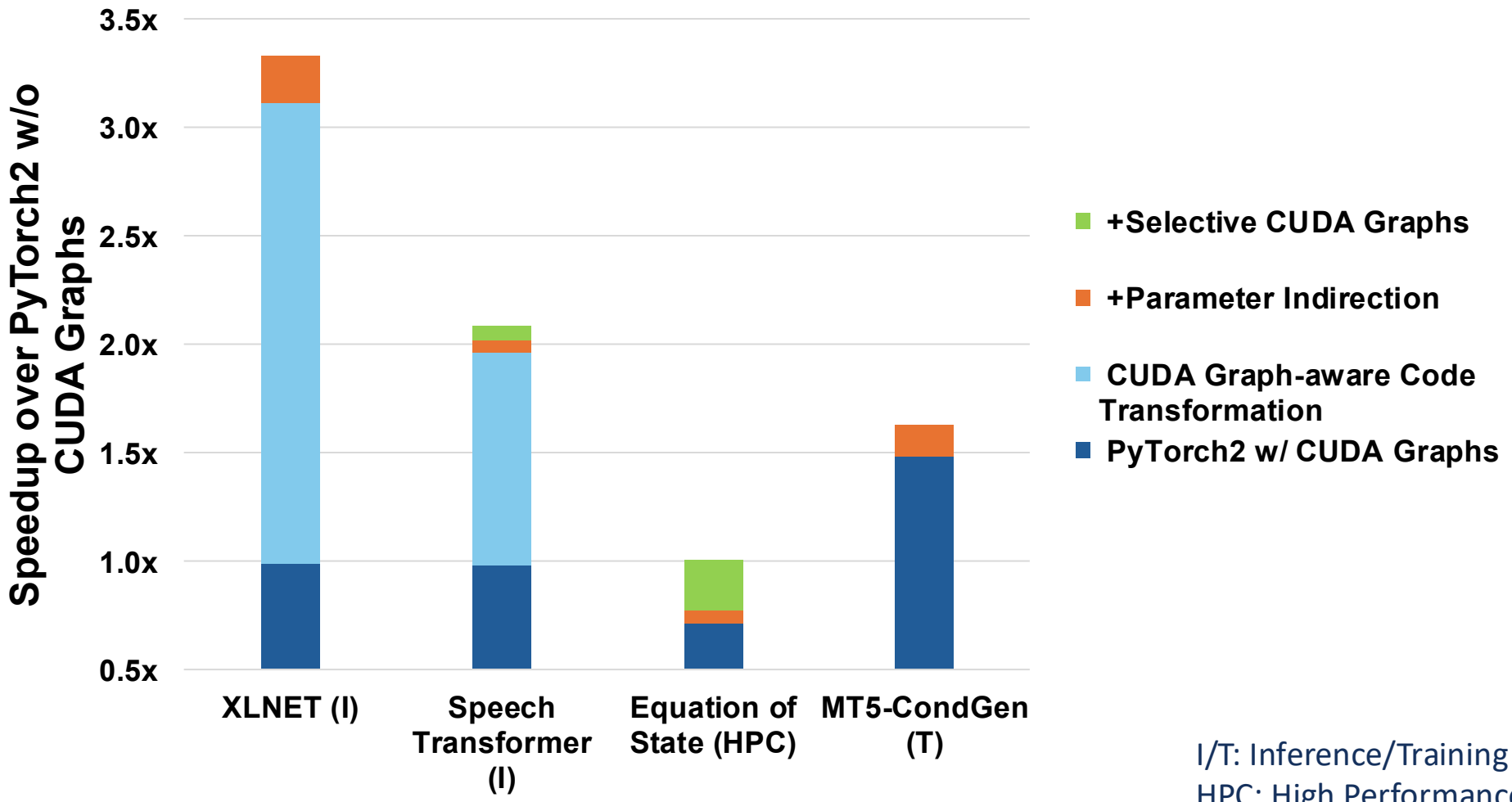
29% geomean speedup over PyTorch2 (w/ CUDA Graphs), up to 3.34x across 25 ML workloads from TorchBench, TIMM, Huggingface

Performance Gains with GraCE



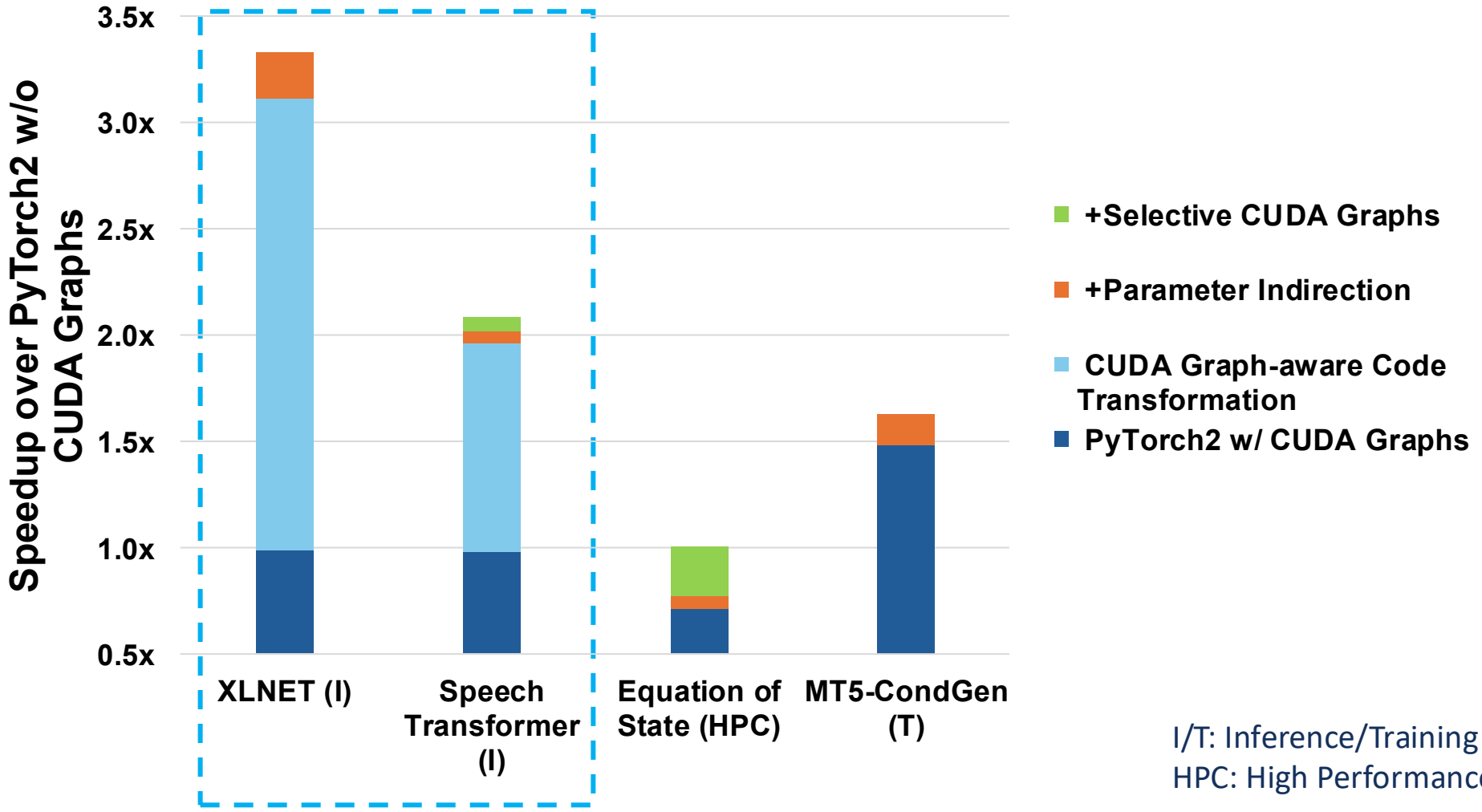
29% geomean speedup over PyTorch2 (w/ CUDA Graphs), up to 3.34x across 25 ML workloads from TorchBench, TIMM, Huggingface

Where Do GraCE's Gains Come From?



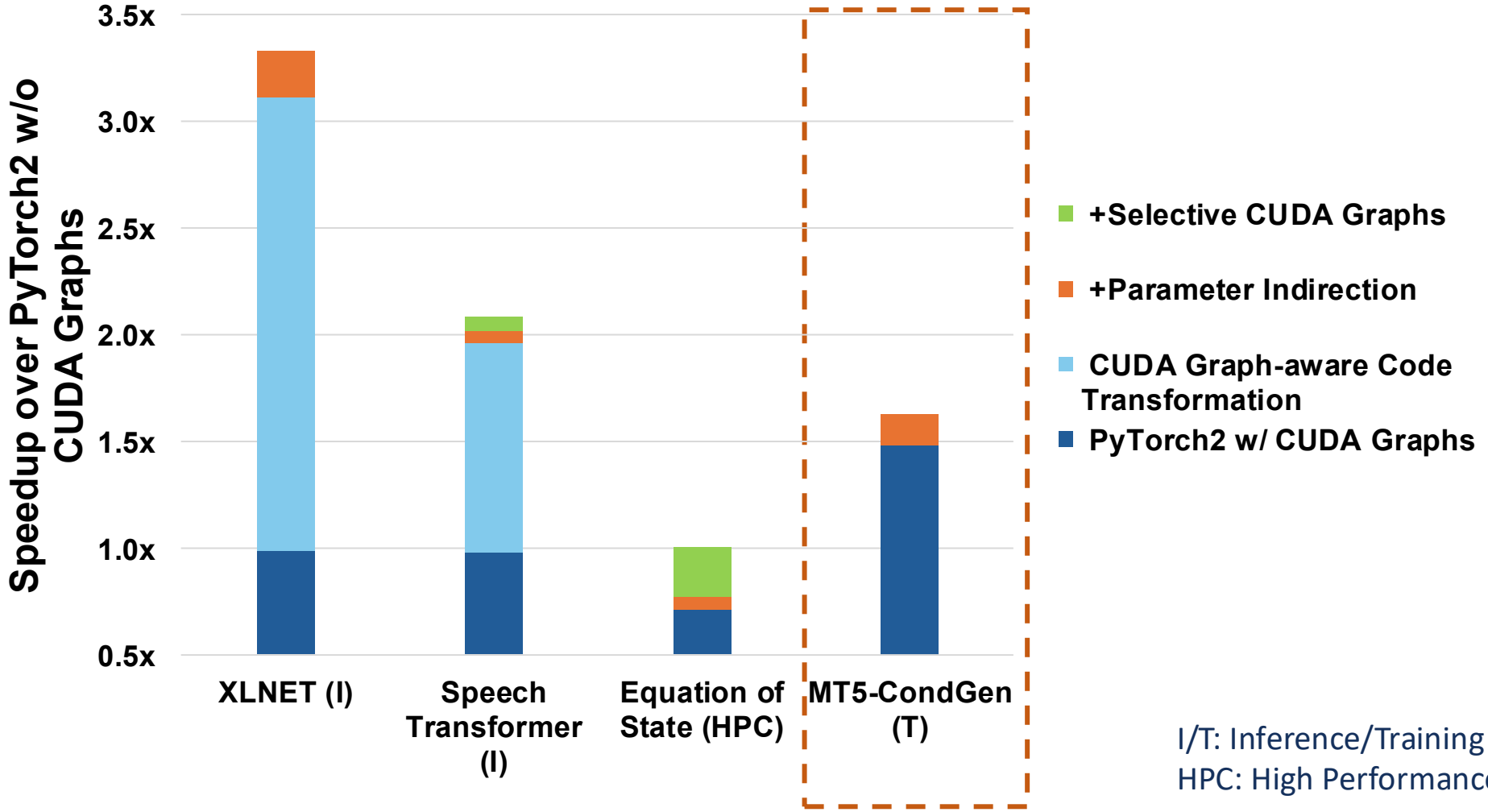
I/T: Inference/Training
HPC: High Performance Computing

Where Do GraCE's Gains Come From?



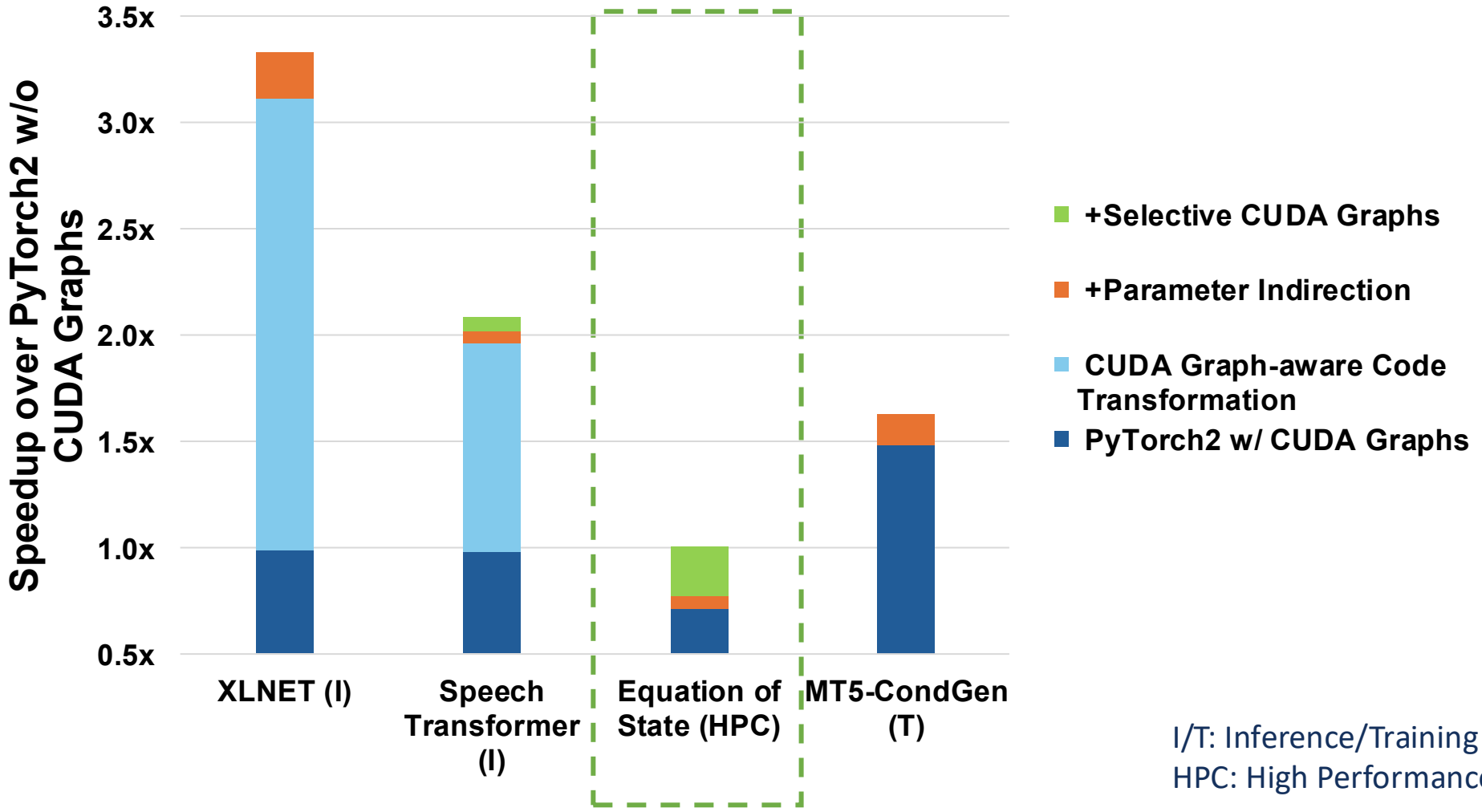
I/T: Inference/Training
HPC: High Performance Computing

Where Do GraCE's Gains Come From?



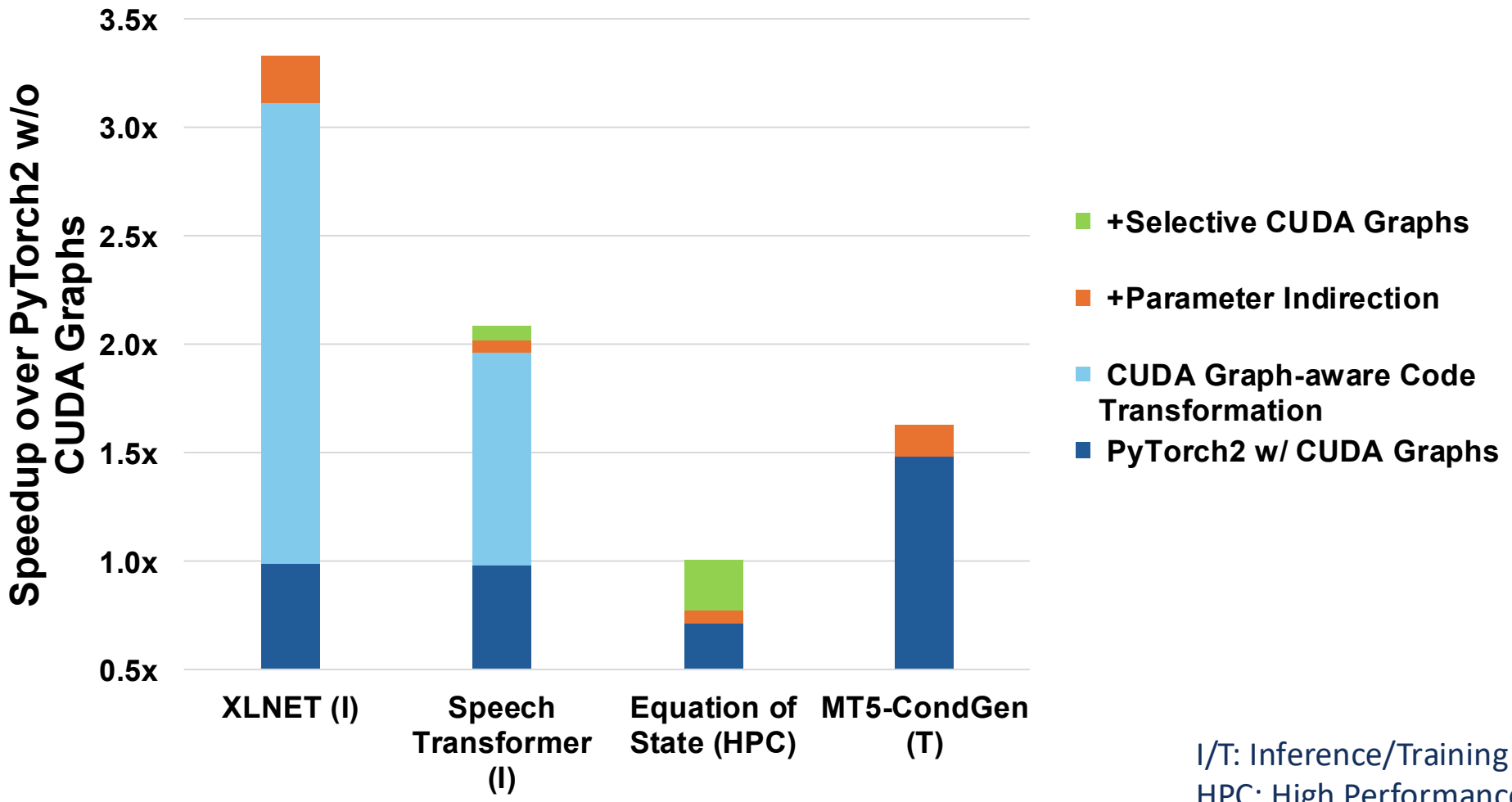
I/T: Inference/Training
HPC: High Performance Computing

Where Do GraCE's Gains Come From?



I/T: Inference/Training
HPC: High Performance Computing

Where Do GraCE's Gains Come From?



I/T: Inference/Training
HPC: High Performance Computing

PyTorch Applications



GraCE

CUDA Graphs



- CUDA Graphs are important but hard to deploy manually
- GraCE: Effective Compiler Support for CUDA Graphs
  - Increases CUDA Graph coverage
  - Reduces replay overhead
  - Selectively deploys CUDA Graph
  - Requires no user-level code changes

29%
geomean speedup

up to 3.34x

Artifact & Code

github.com/csl-iisc/GraCE-OSDI26-Artifact